ARM926EJ-S™

32-bit Microprocessor

NUC980 NuWriter 使用手冊

For additional information or questions, please contact: Nuvoton Technology Corporation.

www.nuvoton.com

Table of Contents

1 概述

NuWriter是NUC980系列的燒錄工具，提供在PC上使用的軟體及其原始碼，使用者可以依照需求自行開發功能。NuWriter工具能幫助使用者透過晶片的USB ISP模式搭配PC上使用的軟體，利用USB設備進行資料傳輸，將Image檔案燒寫至不同的儲存設備(NAND Flash、SPI Flash或eMMC/SD)。新唐科技NUC980系列晶片支援下列四種開機方法：

- eMMC/SD 開機
- SPI Flash 開機
- NAND Flash 開機
- USB ISP 開機

以上四種開機方式是依據Power-on Setting選擇決定。NUC980系列晶片開機時透過Power-on Setting來決定禁用或使能各種功能，例如開機源選擇、QSPI0時鐘源的選擇、看門狗定時器和JTAG介面禁用/使能等功能。Power-on setting可以透過開發板上的指撥開關SW2來設定其對應功能，指撥開關SW2.1~SW2.10設定為ON/OFF分別對應為GPIO PG0~PG9 腳位輸出電位為0/1。Table 1描述每個Power-on Setting位元的定義。

Power-On Setting GPIO PG	開發板指撥開關	描述	Power-On Setting 寄存器
USB0_ID	N/A	**USB端口0角色選擇** 0 = USB端口0作為一個USB主機。 1 = USB端口0作為一個USB設備。	USBID (SYS_PWRON[16])
PG[1:0]	SW2.2/SW2.1	**開機源選擇** 00(ON ON) = USB開機。 01(ON OFF) = eMMC/SD開機 10(OFF ON) = NAND Flash開機。 11(OFF OFF) = SPI Flash開機。	BTSSEL (SYS_PWRON[1:0])
PG.2	SW2.3	**QSPI0時鐘** 0(ON) = QSPI0時鐘30 MHz。 1(OFF) = QSPI0時鐘50 MHz。	QSPI0CKSEL (SYS_PWRON[2])
PG.3	SW2.4	**看門狗定時器 禁用/使能** 0(ON) =在上電之後看門狗定時器禁用。 1(OFF) =在上電之後看門狗定時器使能。	WDTON (SYS_PWRON[3])
PG.4	SW2.5	**JTAG 介面禁用/使能** 0(ON) = PA[6:2] 為通用I/O端口。 1(OFF) = PG[15:11] 為JTAG介面。	JTAGSEL (SYS_PWRON[4])
PG.5	SW2.6	**UART 0 輸出除錯訊息使能/禁用** 0(ON) = UART 0輸出除錯訊息使能 。 1(OFF) = UART 0輸出除錯訊息使能 禁用。	URDBGON (SYS_PWRON[5])
PG[7:6]	SW2.8/ SW2.7	**NAND Flash 的頁大小(Page size)選擇** 00(ON ON) = NAND Flash 頁大小選擇2KB 。	NPAGESEL (SYS_PWRON[7:6])

		01(ON OFF) = NAND Flash頁大小選擇4KB 。 10(OFF ON) = NAND Flash頁大小選擇8KB 。 11(OFF OFF) = 保留。	
PG[9:8]	SW2.10/ SW2.9	**其他配置** 當BTSSEL = 01(ON OFF)時，從SD / eMMC開機，MISCCFG用來定義開機來源從SD0 / eMMC0或是SD1 / eMMC1。 11(OFF OFF) =從SD0 / eMMC0(GPC) 開機。 其他=從SD1 / eMMC1(GPF) 開機。 當BTSSEL = 10(OFF ON), 從NAND Flash開機, MISCCFG用來定義NAND Flash ECC 選擇 00(ON ON) = NAND Flash 自帶 ECC 校驗 。 01(ON OFF) = NAND Flash ECC選擇BCH T12 。 10(OFF ON) = NAND Flash ECC選擇BCH T24 。 11(OFF OFF) = 保留。 當BTSEL = 11時，從SPI Flash開機，MISCCFG定義SPI Flash類型和資料寬度。 00(ON ON) = 資料寬度1-bit SPI-NAND Flash 。 01(ON OFF) = 資料寬度4-bit SPI-NAND Flash 。 10(OFF ON) =資料寬度4-bit SPI-NOR Flash 。 11(OFF OFF) =資料寬度1-bit SPI-NOR Flash 。	MISCCFG (SYS_PWRON[9:8])

Table 1 Power On Setting

1.1 NAND 開機

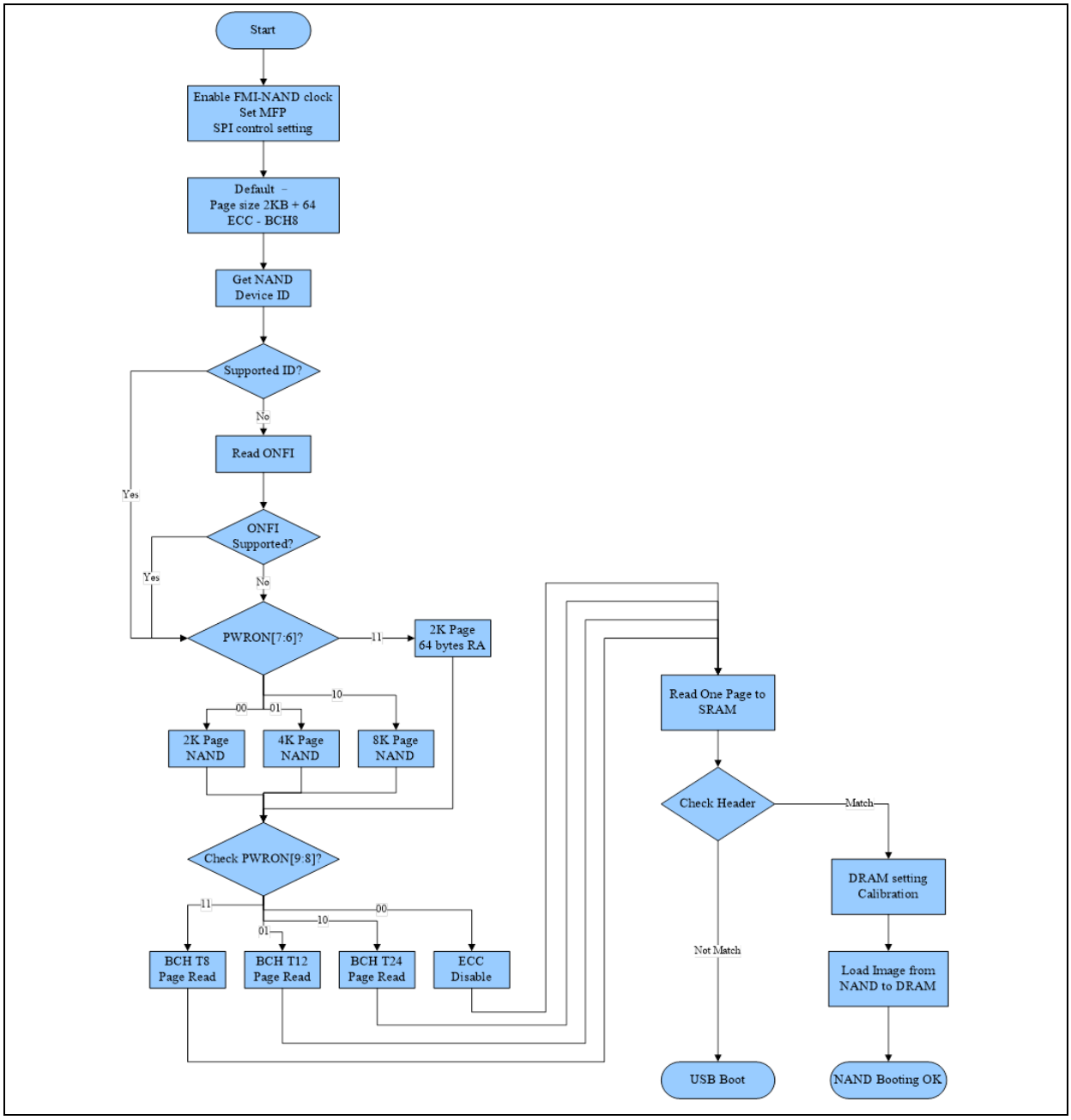


Figure 1-1 Nand Startup Flow

NAND Flash開機流程偵測PWRON[7:6]確定Page Size，接著偵測PWRON[9:8]來確定NAND Flash ECC Type，之後直接讀取NAND Flash中Block0確定檔頭中的Boot Code Marker(參考4.3.1.1章節)是否正確，如果不正確繼續往下一個Block來確認檔頭中的Boot Code Marker是否正確，當Block0、Block1、Block2、Block3都找不到檔頭(Header)中的Boot Code Marker時，則跳回到USB開機。當偵測檔頭資料正確時，接下來進行DDR初始化參數設定，最後會將剛才讀取到的Loader Image複製到對應的DDR的位置，並且跳過去執行，即完成NAND開機的流程。注意：Loader 中存放Image加上檔頭和DDR參數的總長度不能超過一個Block大小，即Loader Image + Header + DDR Parmater ≤ Block Size 。

1.2 SPI 開機

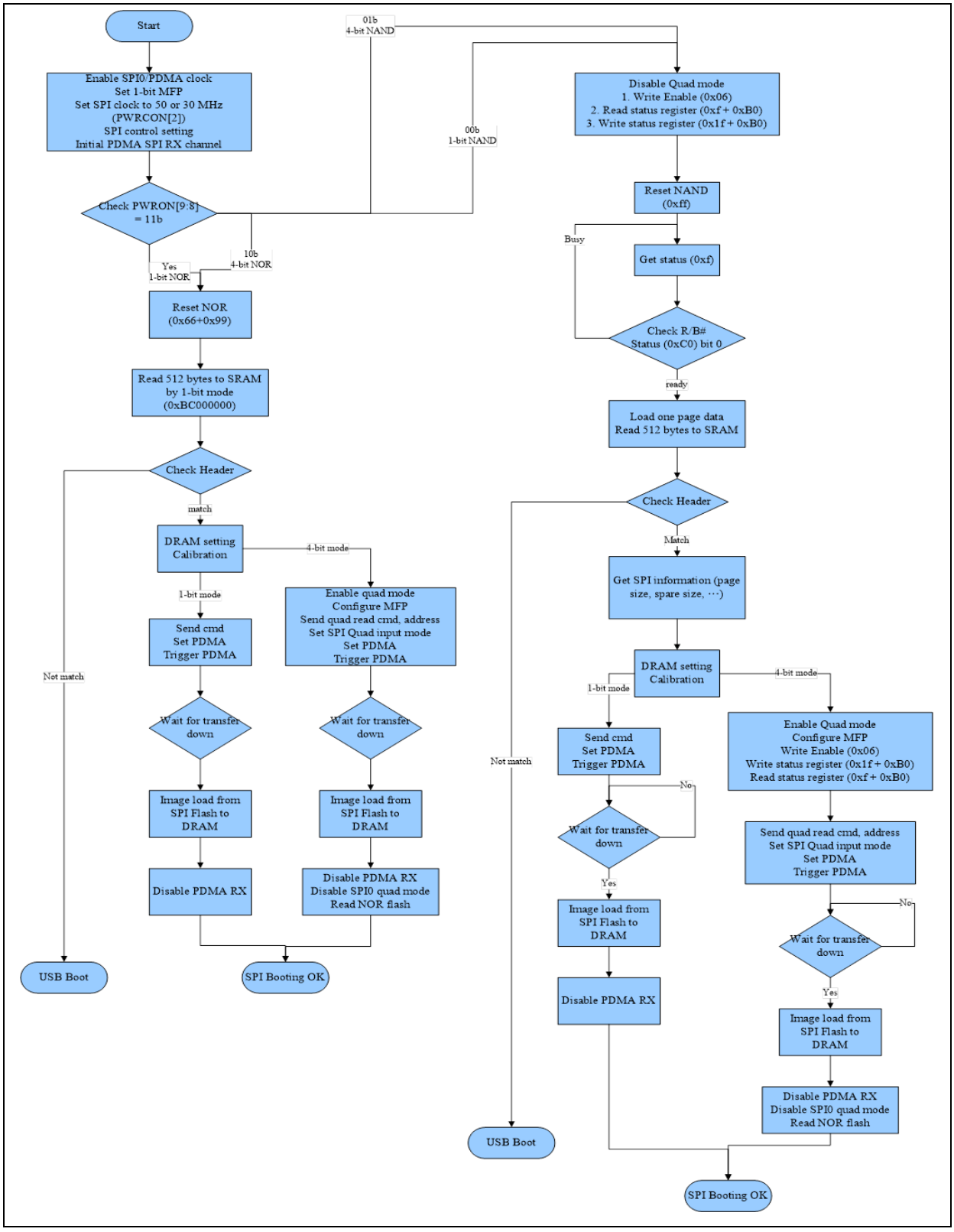


Figure 1-2 SPI Startup Flow

SPI開機後直接讀取SPI Flash中0x0000000位置來確定檔頭中的Boot Code Marker(參考4.4.1.1章節)是

否正確，如果不正確則跳到USB開機。當檔頭中的Boot Code Marker偵測正確，接下來進行DDR初始化參數設定，最後會將剛才讀取到的Loader Image複製到對應的DDR的位置，並且跳過去執行，即完成SPI開機的流程。

1.3 eMMC/SD 開機

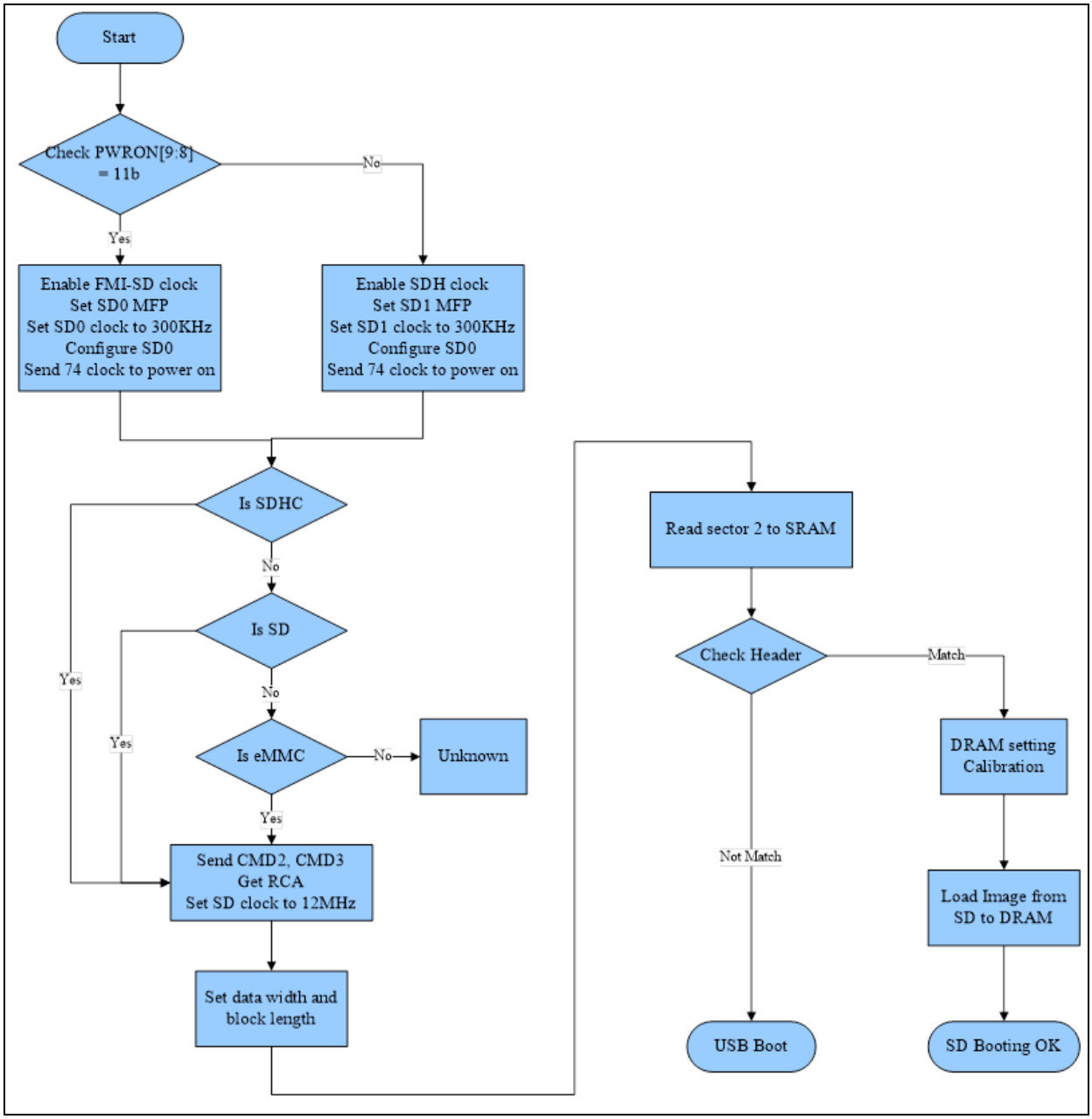


Figure 1-3 eMMC/SD Startup Flow

eMMC/SD 開機後偵測PWRON[9:8]確定是eMMC0/SD0還是eMMC1/SD1，直接讀取eMMC中0x00000400位置來確定檔頭中的Boot Code Marker(參考4.5.1.1章節)是否正確，如果不正確則跳到USB開機。當檔頭中的Boot Code Marker偵測正確，接下來進行DDR初始化參數設定，最後會將剛才讀取到的Loader Image複製到對應的DDR的位置，並且跳過去執行，即完成eMMC/SD開機的流程。

1.4 USB 開機

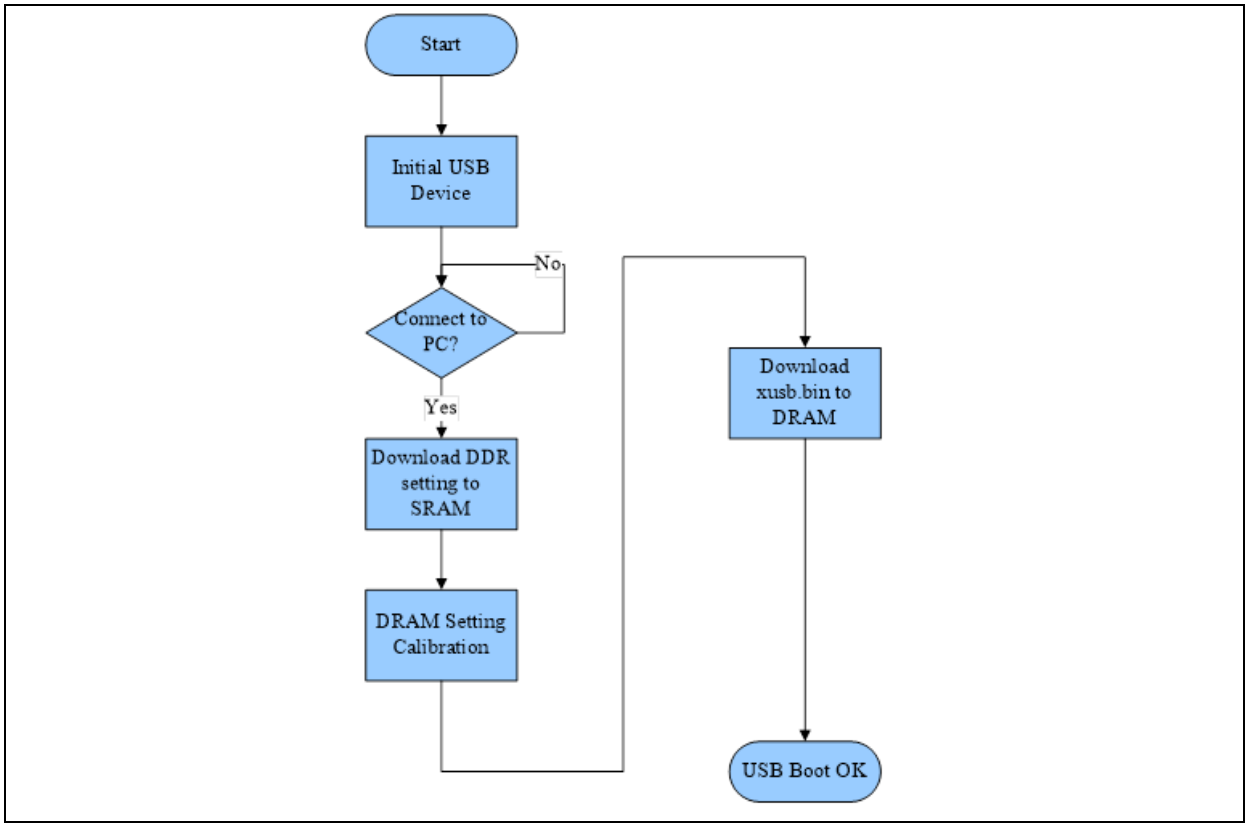


Figure 1-4 USB Startup Flow

USB開機後第一個動作會等待DDR參數初始化完成，所以會等待PC端的NuWriter連上NUC980並且等待NuWriter傳送DDR參數做初始化，完成後馬上會再傳送xusb.bin並且跳過去執行，此時xusb.bin運作起來會與PC端的NuWriter溝通。

2 安裝USB驅動程式

NuWriter必須在電腦中安裝USB VCOM驅動程式才能使用NuWriter工具，USB VCOM驅動程式的安裝方式請依照下列步驟進行：

步驟1：將電腦與NUC980系列晶片透過USB cable連接後，開啟NUC980系列晶片的電源之後，Windows會發現新的USB設備，在電腦端選取執行檔*WinUSB4NuVCOM.exe*開始安裝驅動程式(Figure 2-1)。

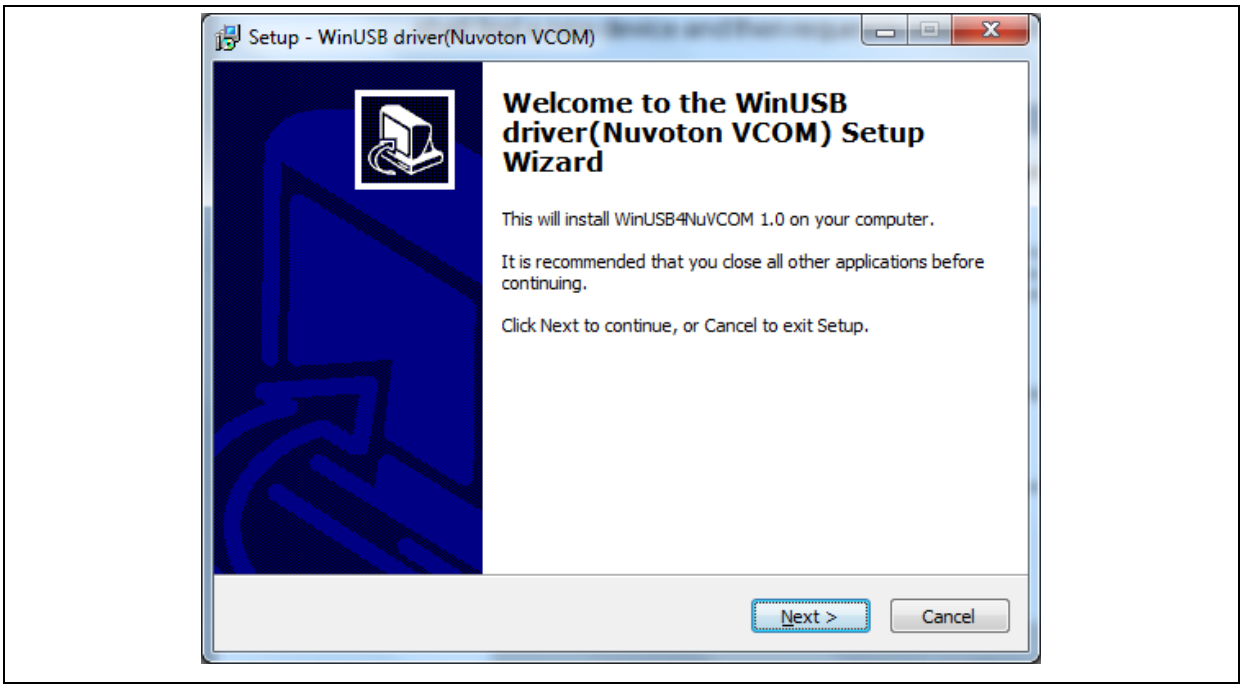


Figure 2-1 WinUSB4NuVCOM Driver Setup (1)

步驟2：按下 “**Next**”。這個畫面告訴你即將要安裝WinUSB4NuVCOM 1.0 驅動程式(Figure 2-2)。

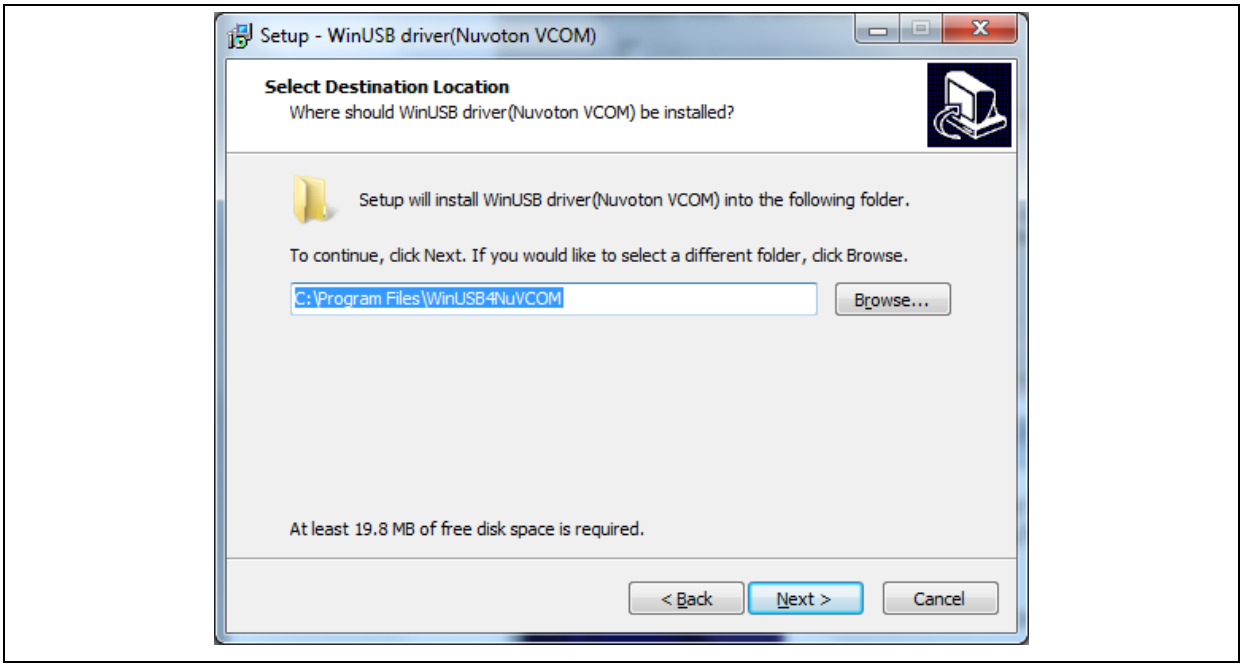


Figure 2-2 WinUSB4NuVCOM Driver Setup (2)

步驟3：選擇使用者想要安裝的路徑或使用預設的路徑，確定後按下“**Next**”(Figure 2-3)。

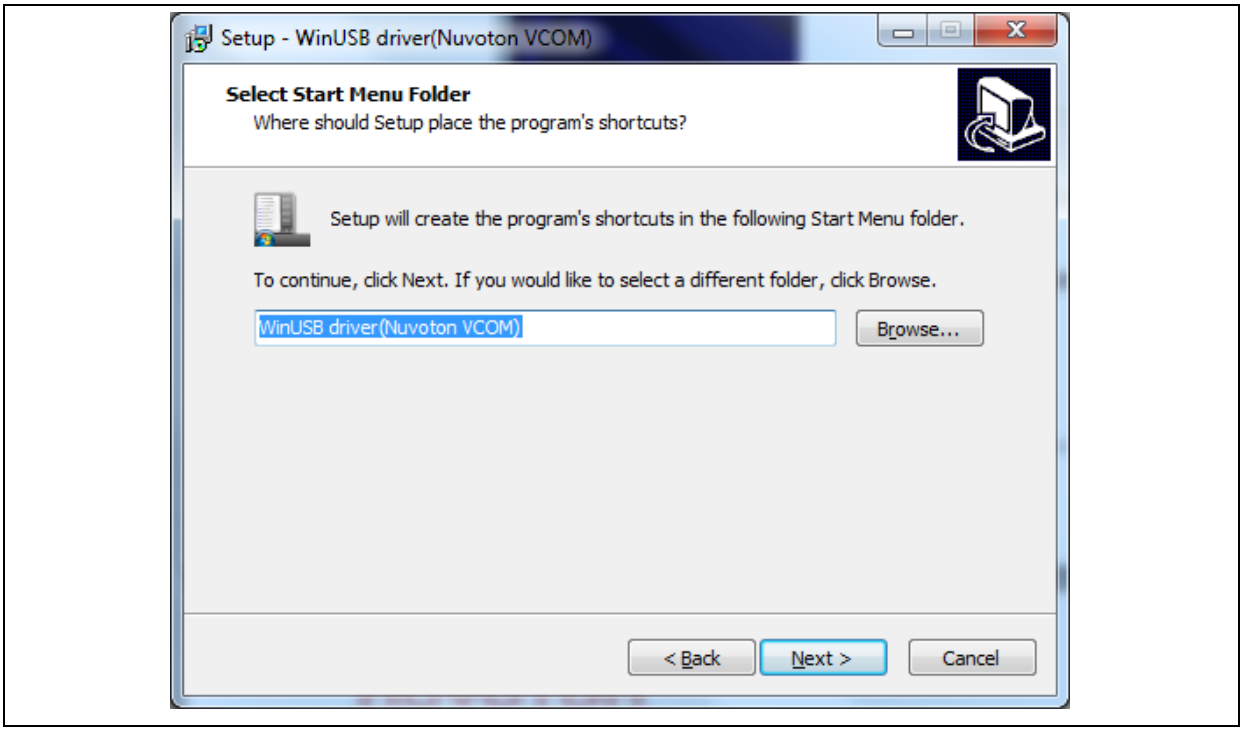


Figure 2-3 WinUSB4NuVCOM Driver Setup (3)

步驟4：按下 “**Next**”(Figure 2-4)。

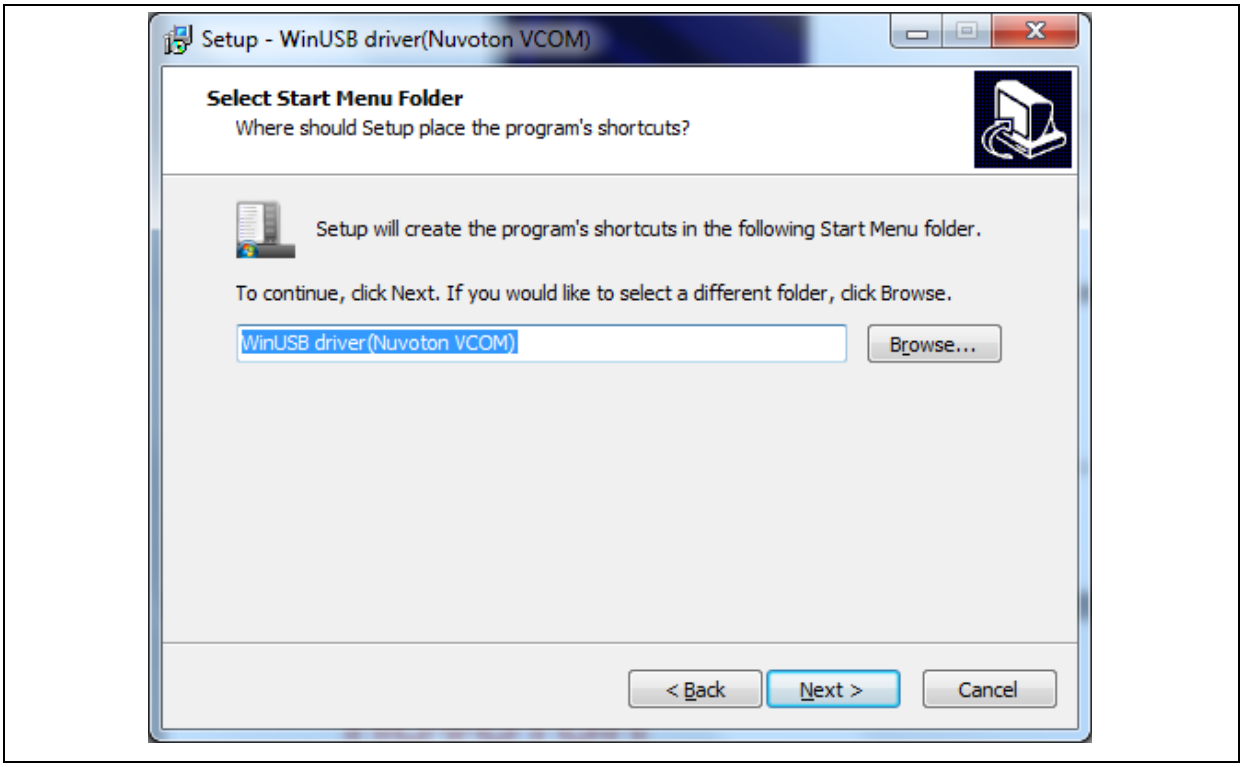


Figure 2-4 WinUSB4NuVCOM Driver Setup (4)

步驟5：按下 “**Install**”(Figure 2-5)。

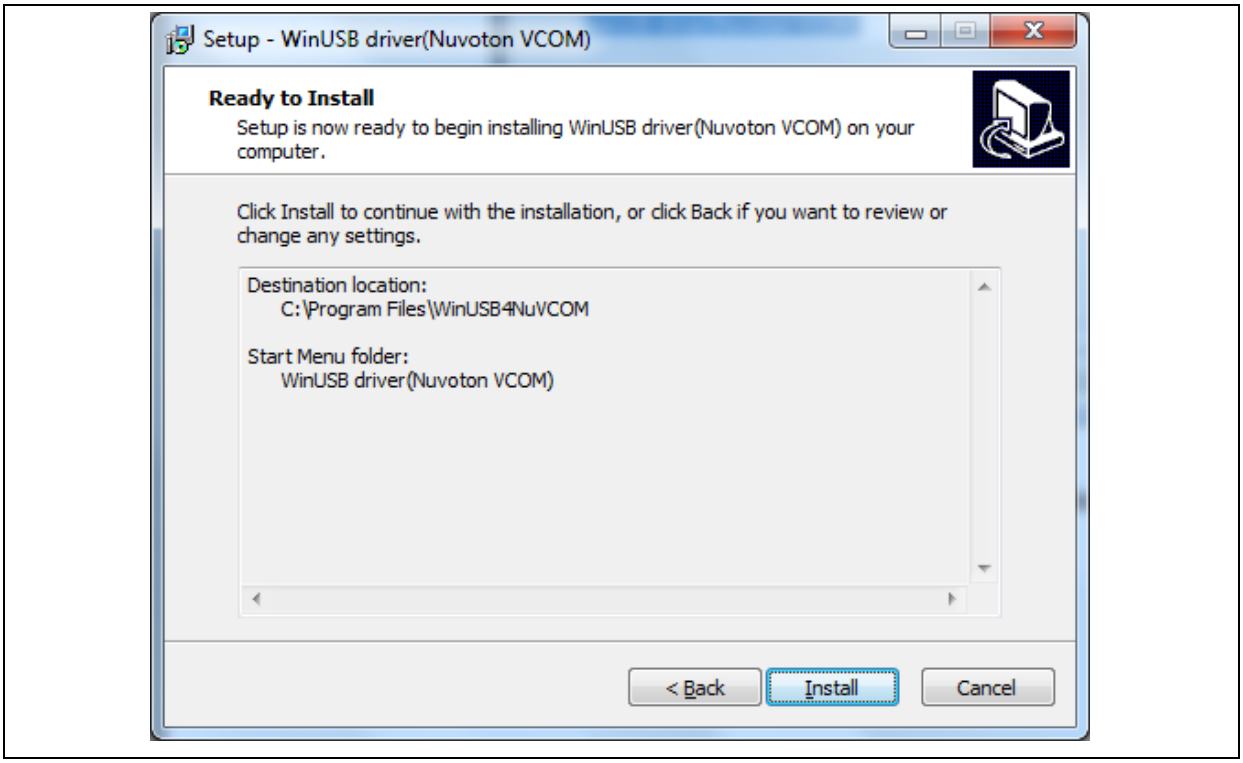


Figure 2-5 WinUSB4NuVCOM Driver Setup (5)

步驟6：按下 “**Finish**”，完成USB VCOM驅動程式的安裝(Figure 2-6)。

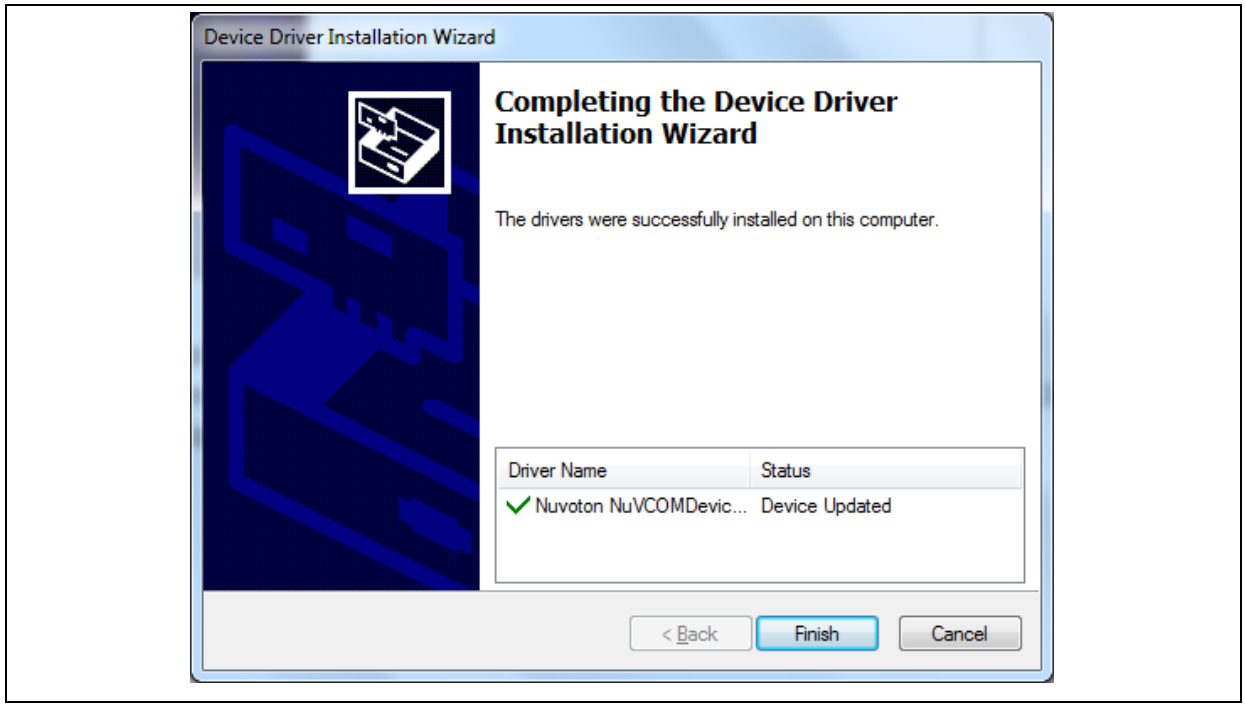


Figure 2-6 WinUSB4NuVCOM Driver Setup (6)

安裝USB VCOM驅動程式順利完成後，使用者可自行確認此時的Windows作業系統應該會自行偵測到新的裝置，並進而自動載入其相對應的USB設定。使用者可於Device Manager中看到“WinUSB driver (Nuvoton VCOM)”，即代表驅動程式成功安裝完畢(Figure 2-7)。

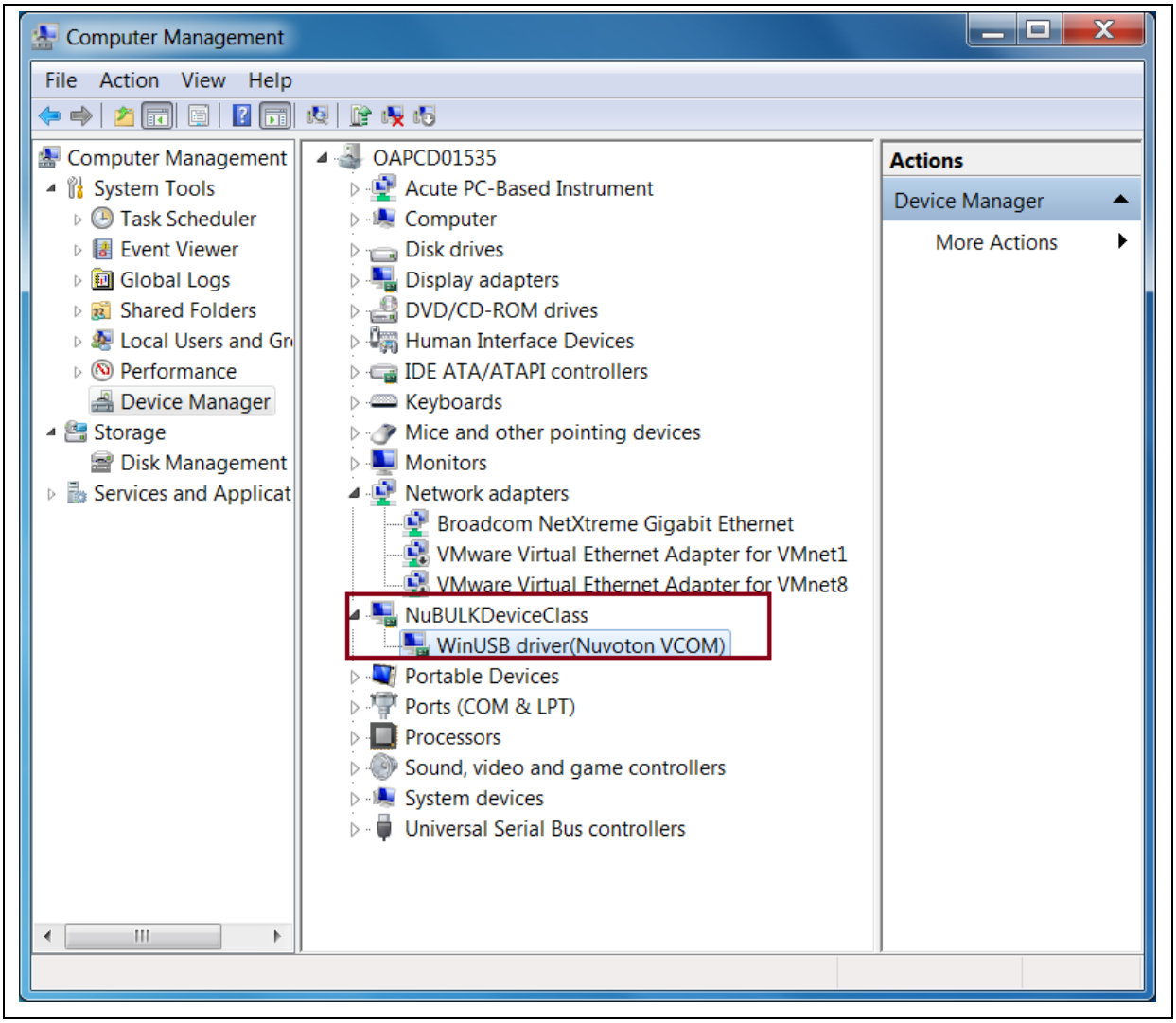


Figure 2-7 Nuwriter VCOM Device

3 USB ISP 模式

NUC980系列晶片開機時透過Power-on Setting去選擇開機的方法，Power-on Setting可以透過開發板上的指撥開關SW2.1~SW2.10設定相關功能。舉例來說，選擇USB ISP模式，開發板上指撥開關SW2.1及SW2.2都設定為On，其他開機設定請參考下表：

開機設定	SW2.2	SW2.1
USB ISP 開機	On	On
eMMC/SD 開機	On	Off
NAND 開機	Off	On
SPI 開機	Off	Off

開發板上的Power-on Setting指撥開關SW2.1和SW2.2設定為USB ISP模式，使用者可於Tera Term終端機輸出結果看到Boot from USB(Figure 3-1)。

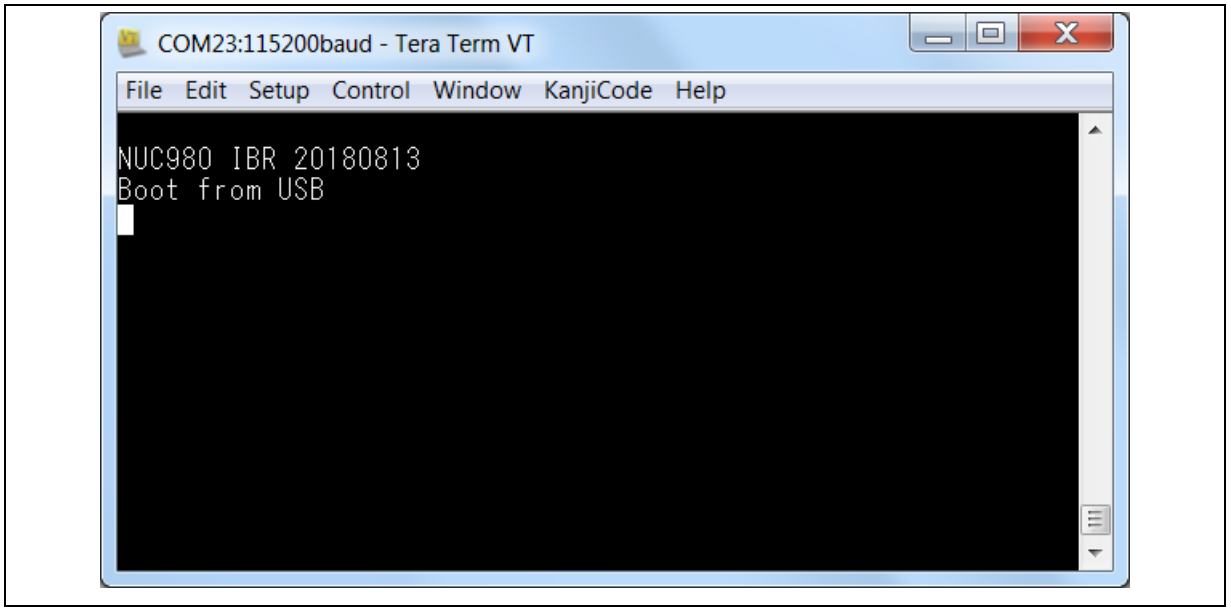


Figure 3-1 NuWriter USB ISP Mode

4 軟體介面及功能說明

開啟電腦端的NuWriter工具執行“*NuWriter.exe*”，選擇目標晶片(NUC980 series)後再依照晶片型號(NUC980DF61YC.ini)選擇DDR初始化參數(Figure 4-1)，完成後按下“**Continue**”。當Nuwriter連線成功時，使用者可以在終端機看到開發板上所有儲存裝置的資訊並顯示”Finish get INFO”(Figure 4-2)，即可開始使用。注意：如果電腦沒有找到WinUSB4NuVCOM驅動程式則NuWriter工具無法使用。

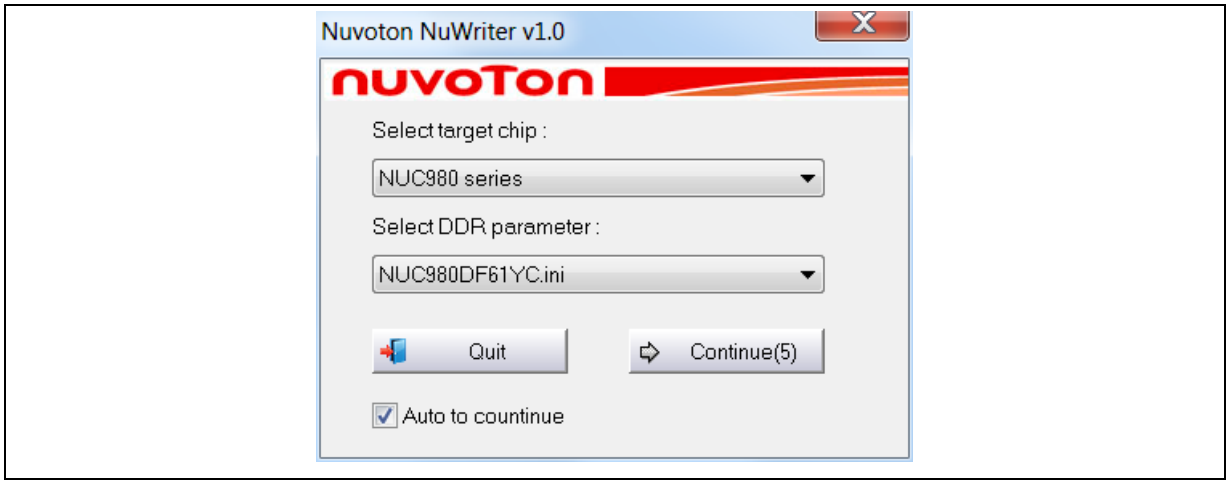


Figure 4-1 NuWriter Set Chip/DDR

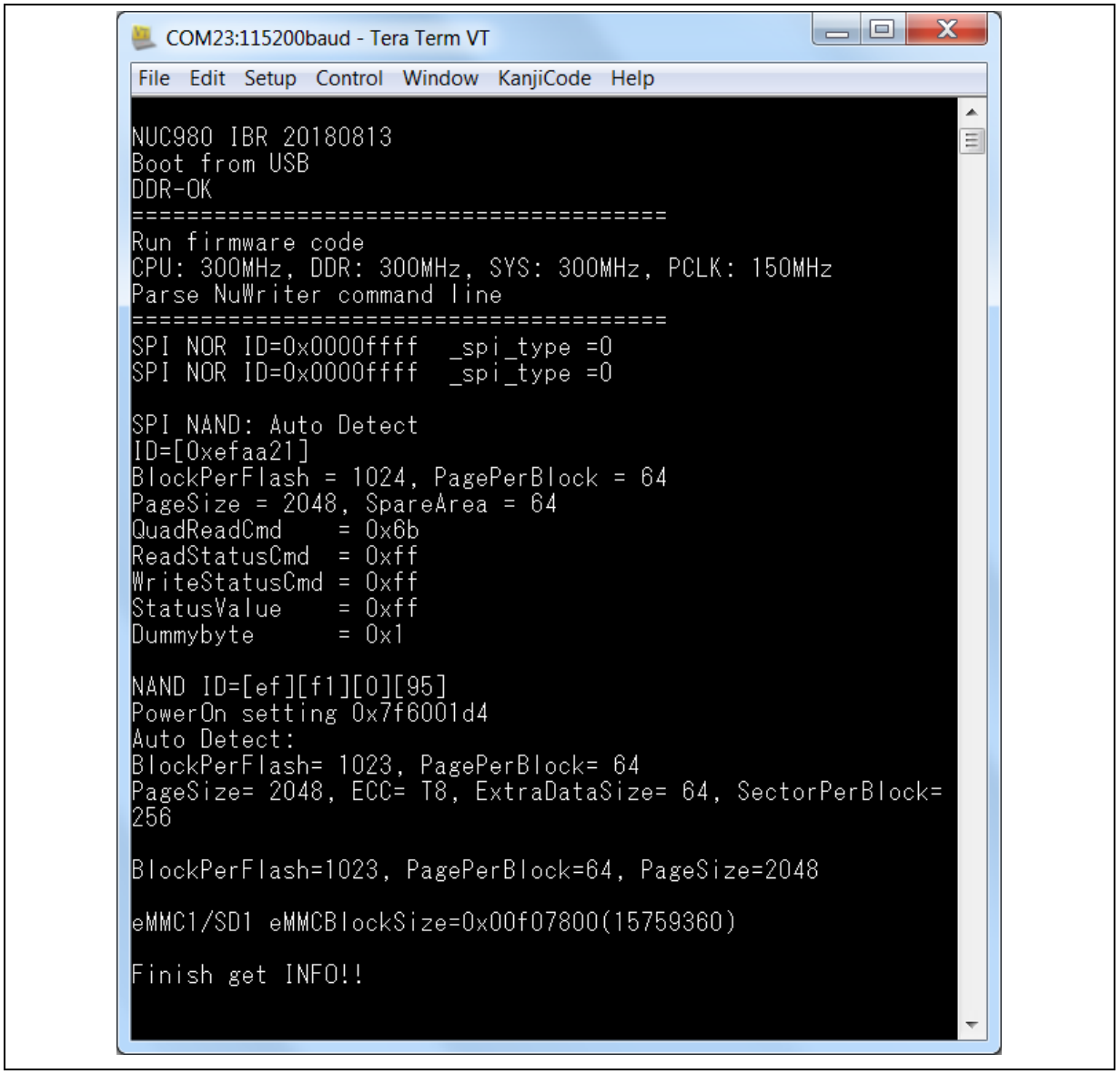


Figure 4-2 NuWriter Console Message

4.1 操作視窗

使用者可以從Nuwriter軟體視窗分辨PC Tool版號、FW版號及DDR參數版號(Figure 4-3)。軟體操作視窗由Flash Type下拉式選單、顯示Connection State按鈕，加上Function Control視窗三個部分組合而成(Figure 4-4)。Flash Type下拉式選單讓使用者選擇Flash Type，再經由下方的Function Control視窗透過PC去設定各種Flash所要燒錄的功能；Connection State按鈕是用來偵測Nuwriter的USB裝置，判斷USB驅動程式是否安裝成功。

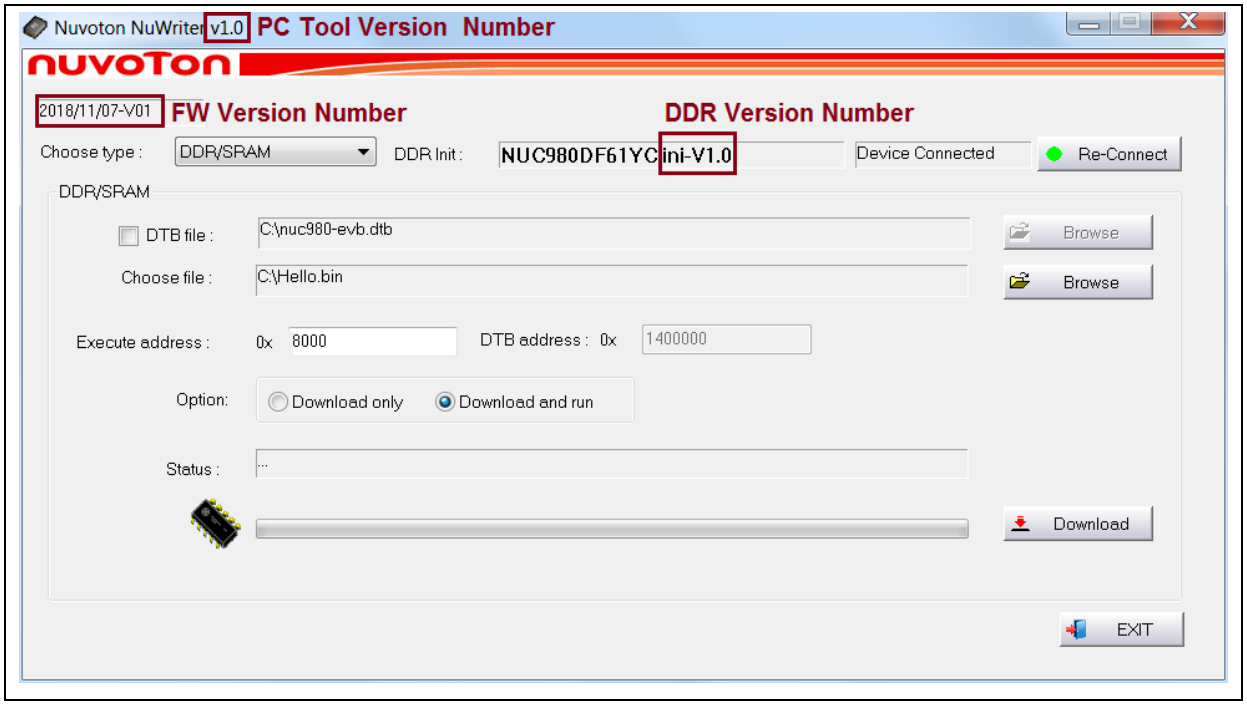


Figure 4-3 Nuwriter Version Number

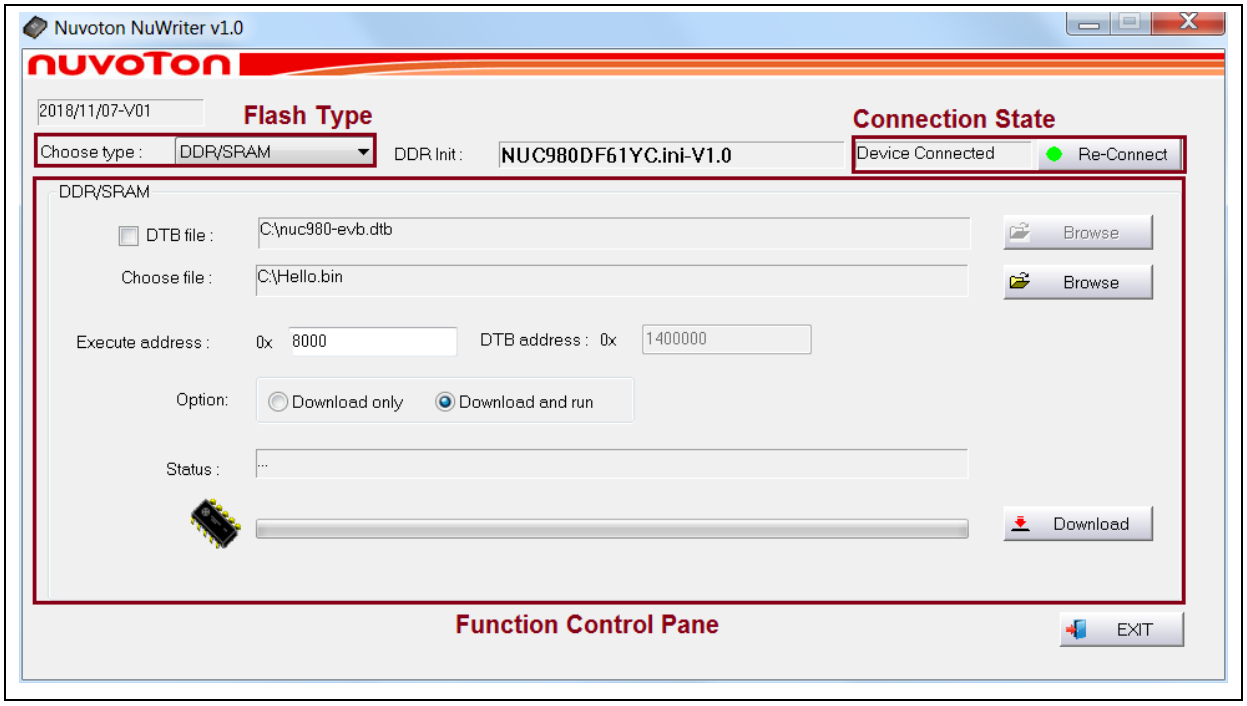


Figure 4-4 Nuwriter Operating Window

4.2 DDR/SRAM 模式

NUC980晶片提供Linux Kernel和無作業系統的BSP，使用者可以使用DDR/SRAM模式燒寫至記憶體內執行。另外，Linux內核 Device tree使用的DTB檔案也可以使用該模式燒寫。

4.2.1 操作方法

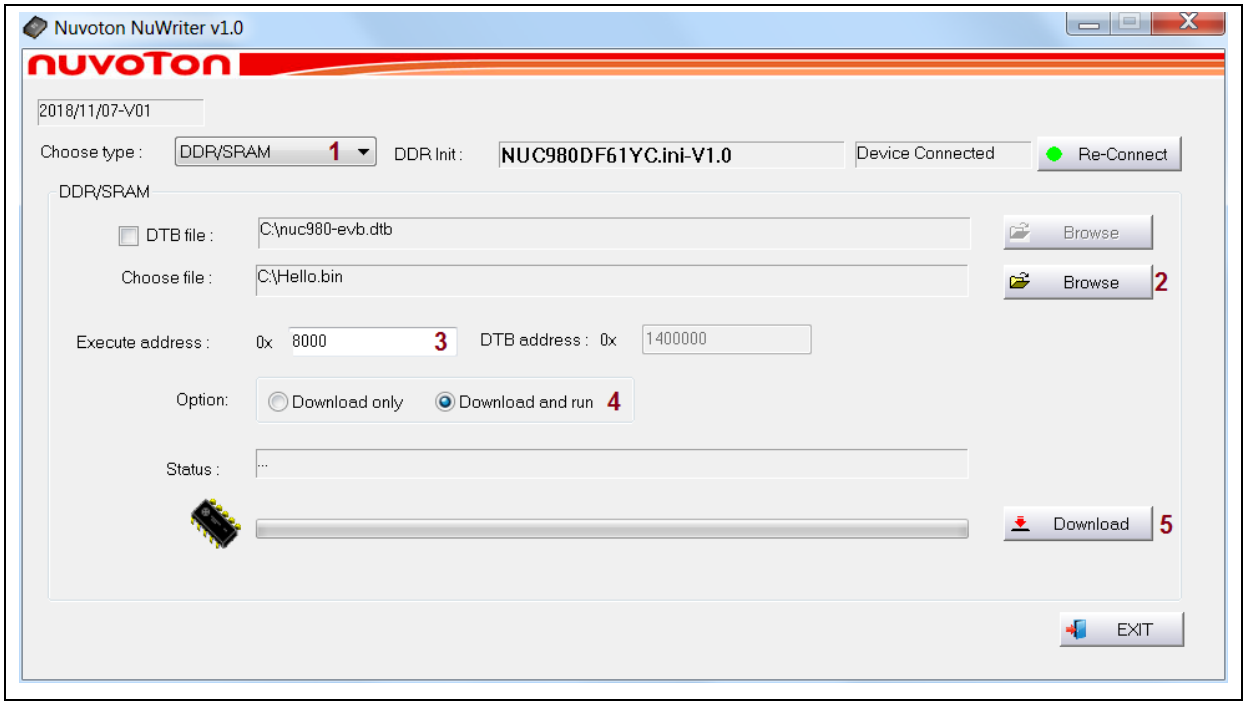


Figure 4-5 NuWriter – DDR/SDRAM Mode (1)

依照Figure 4-5步驟即可在DDR/SRAM模式下，將Image檔案直接下載到DDR或SRAM記憶體中。操作步驟說明如下：

1. 選擇 “**DDR/SRAM**”模式。
2. 選擇Image檔案。
3. 輸入Image檔案放在DDR/SRAM的位址。注意：若要傳輸到DDR中，不論型號的DDR，最後 1MB 不可用。
4. 選擇”**Download only**”或是選擇”**Download and run**”，此選項會在下載後直接執行。
5. 按下 “**Download**”。

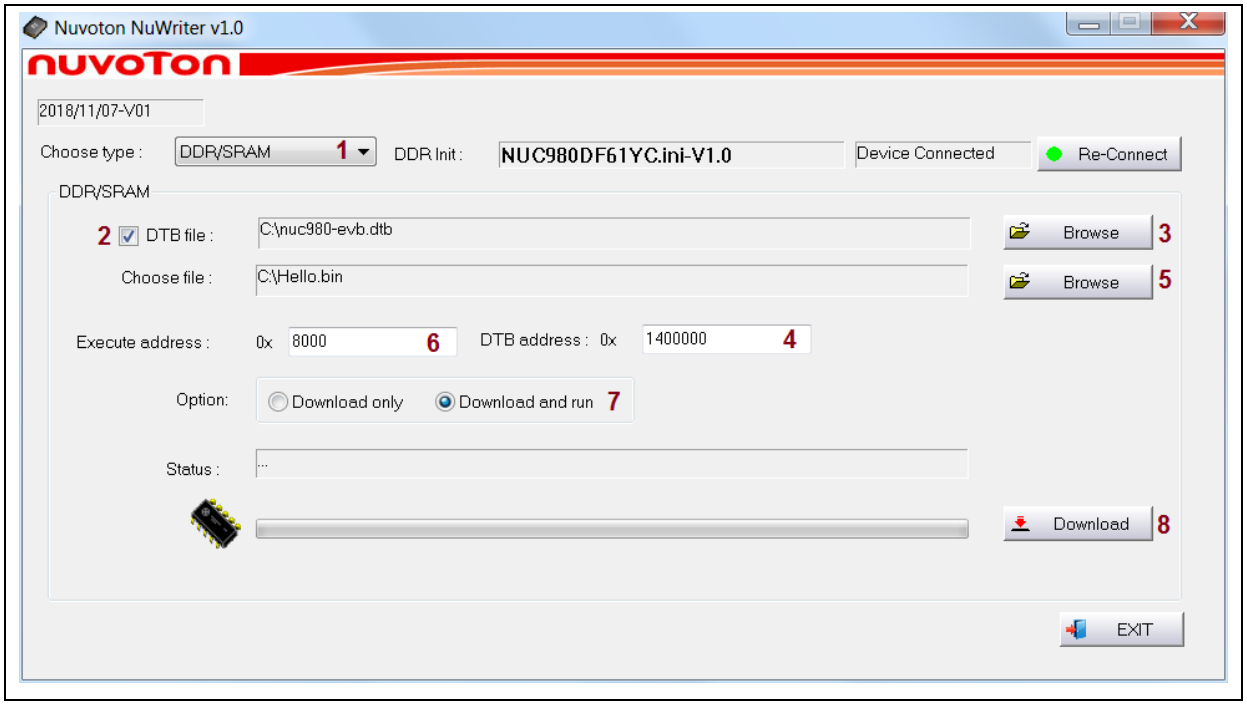


Figure 4-6 NuWriter – DDR/SDRAM Mode (2)

依照Figure 4-6步驟即可以在DDR/SRAM模式下，將Image檔案和Linux Device tree使用的DTB檔案一起下載至DDR或SRAM記憶體中，操作步驟如下：

1. 選擇 “**DDR/SRAM**”模式。
2. 勾選DTB
3. 選擇 DTB檔案
4. 輸入DTB檔案放在DDR/SRAM的位址. 注意：不要被Image 檔案覆蓋.
5. 選擇Image檔案。
6. 輸入Image檔案放在DDR/SRAM的位址。注意：若要傳輸到DDR中，不論型號的DDR，最後 1MB 不可用。
7. 選擇”**Download only**”或是選擇”**Download and run**”，此選項會在下載後直接執行。
8. 按下 “**Download**”。

4.3 NAND Flash模式

NAND模式可以將Image檔案燒寫至NAND Flash，Image檔案型態為Loader、Data、Environment、Pack四種型態中的其中一種。

4.3.1 Image檔案型態

4.3.1.1 Loader 型態

Loader 主要是作為開機的第一支程式，Loader分成Main U-Boot和SPL U-Boot兩個部分，Main U-Boot是完整功能的U-Boot，SPL U-Boot將Main U-Boot從NAND/SPI NAND Flash搬到 DDR 執行。SPL U-Boot只有NAND和SPI NAND boot時才會用到；SPI 開機或eMMC/SD 開機只需要Main U-Boot。SPL U-Boot 的鏈結位址預設為0x200，Main U-Boot的鏈結位址是0xE00000。當NUC980系列晶片上電後會讀取Loader型態的Image並執行，不限定一定要存放Loader，一般程式也是可以使用Loader型態來作為上電後第一支執行程式。Loader Image增加32bytes的檔頭(Header)和DDR參數合併而成，Figure 4-7為Loader型態的Image格式。

	0x0		0x4				0x8				0xC
0x00	Boot Code Marker		Execute Address				Image Size				Reserved
0x10	Page Size	Spare Area	Quad Read cmd	Read Status cmd	Write Status cmd	Status Value	Dummy Byte	Reserved	Reserved	Reserved	Reserved
0x20	DDR - Initial Marker		DDR - Counter				DDR - Address 0				DDR – Value0
0x30	DDR - Address 1		DDR - Value 1				DDR - Address 2				DDR - Value 2
0x40	DDR - Address 3		DDR - Value 3				DDR - Address 4				DDR - Value 5
0x50	DDR - Address 5		DDR - Value 5				DDR - Dummy				DDR - Dummy
0x60	uBoot										

Figure 4-7 Boot Code Header

Boot Code Marker = {0x20,'T','V','N'} 。

Execute Address = 透過IBR 將Loader 複製到{ Execute Address } 位置。

Image Size = {Loader 檔案長度}。

Page Size = {SPI NAND Flash 頁大小}。

Spare Area = { SPI NAND Flash Spare 大小}。

Quad Read cmd = { SPI Flash Quad Read 指令}。

Read Status cmd = { SPI Flash Read Status 指令}。

Write Status cmd = { SPI Flash Write Status 指令}。

Status Value = { SPI Flash Status 值}。

Dummy Byte = { SPI Flash Dummy Byte 大小}。

DDR - Initial Marker = {0x55AA55AA}。

DDR - Counter = DDR參數的長度，是計算Figure 4-8所選擇DDR參數NUC980DF61Y.ini，可以在NuWriter\sys_cfg\中找到此檔案。

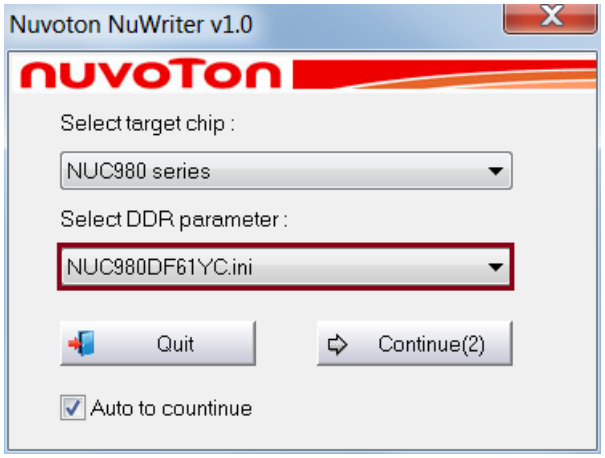


Figure 4-8 NuWriter – Select DDR Parameter

Figure 4-9 *NUC980DF61YC.ini* 存放的是 DDR 的參數。

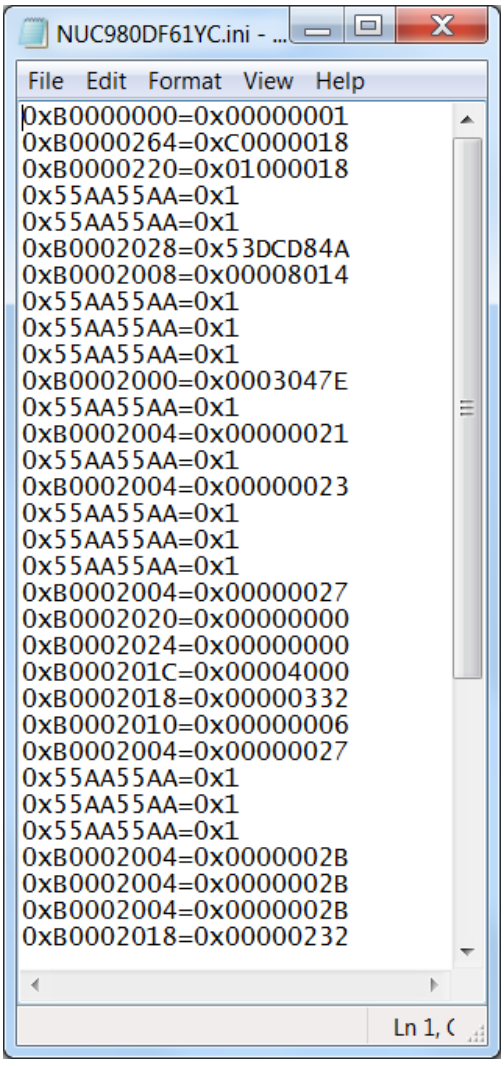


Figure 4-9 NuWriter - NUC980DF61YC.ini

*NUC980DF61YC.ini*內容如下：(實際DDR參數如上圖所示，不需要加入行數)

```
line 1 : 0xB0000000=0x00000001
line 2 : 0xB0000264=0xC0000018
line 3 : 0xB0000220=0x01000018
line 4 : 0x55AA55AA=0x1
line 5 : 0x55AA55AA=0x1
line 6 : 0xB0002028=0x53DCD84A
line 7 : 0xB0002008=0x00008014
line 8 : 0x55AA55AA=0x1
line 9 : 0x55AA55AA=0x1
line 10: 0x55AA55AA=0x1
line 11: 0xB0002000=0x0003047E
line 12: 0x55AA55AA=0x1
line 13: 0xB0002004=0x00000021
line 14: 0x55AA55AA=0x1
line 15: 0xB0002004=0x00000023
line 16: 0x55AA55AA=0x1
line 17: 0x55AA55AA=0x1
line 18: 0x55AA55AA=0x1
line 19: 0xB0002004=0x00000027
line 20: 0xB0002020=0x00000000
line 21: 0xB0002024=0x00000000
line 22: 0xB000201C=0x00004000
line 23: 0xB0002018=0x00000332
line 24: 0xB0002010=0x00000006
line 25: 0xB0002004=0x00000027
line 26: 0x55AA55AA=0x1
line 27: 0x55AA55AA=0x1
line 28: 0x55AA55AA=0x1
line 29: 0xB0002004=0x0000002B
line 30: 0xB0002004=0x0000002B
line 31: 0xB0002004=0x0000002B
line 32: 0xB0002018=0x00000232
line 33: 0xB000201C=0x00004781
line 34: 0xB000201C=0x00004401
line 35: 0xB0002004=0x00000020
line 36: 0xB0002034=0x00888828
line 37: 0x55AA55AA=0x1
line 38: 0x55AA55AA=0x1
line 39: 0x55AA55AA=0x1
line 40: 0x55AA55AA=0x1
```

```
line 41: 0x55AA55AA=0x1
line 42: 0x55AA55AA=0x1
line 43: 0x55AA55AA=0x1
line 44: 0x55AA55AA=0x1
line 45: 0x55AA55AA=0x1
line 46: 0x55AA55AA=0x1
line 47: 0x55AA55AA=0x1
```

依據 *NUC980DF61YC.ini*內容可以計算出 DDR - Counter = 46，line 1為DDR的版號，不列入DDR-Counter。

line 1 DDR - Address = {0xB0000000}。DDR – Value = {0x00000001}。DDR版號為V1.0。

line 2 DDR - Address 0 = {0xB0000264}。DDR – Value0 = {0xC0000018}。

line 3 DDR - Address 1 = {0xB0000220}。DDR – Value1 = {0xC0000018}。

line 4 DDR - Address 2 = {0x55AA55AA}。DDR – Value2 = {0x1}。

line 5 DDR - Address 3 = {0x55AA55AA}。DDR – Value3 = {0x1}。

line 6 DDR - Address 4 = {0xB0002028}。DDR – Value4 = {0x53DCD84A}。

以此類推 ・・・

line 44 DDR - Address 43 = {0x55AA55AA}。DDR – Value43 = {0x1}。

line 45 DDR - Address 44 = {0x55AA55AA}。DDR – Value44 = {0x1}。

line 46 DDR - Address 45 = {0x55AA55AA}。DDR – Value45 = {0x1}。

line 47 DDR - Address 46 = {0x55AA55AA}。DDR – Value46 = {0x1}。

DDR參數存放在儲存體中必須以16byts對齊・不足的部分需要以Dummy = {0x00000000}補足。

U-Boot = 存放Loader binary file。

再燒入到Nand Flash 中Block0，Block1，Block2，Block3，Figure 4-10為示意圖：

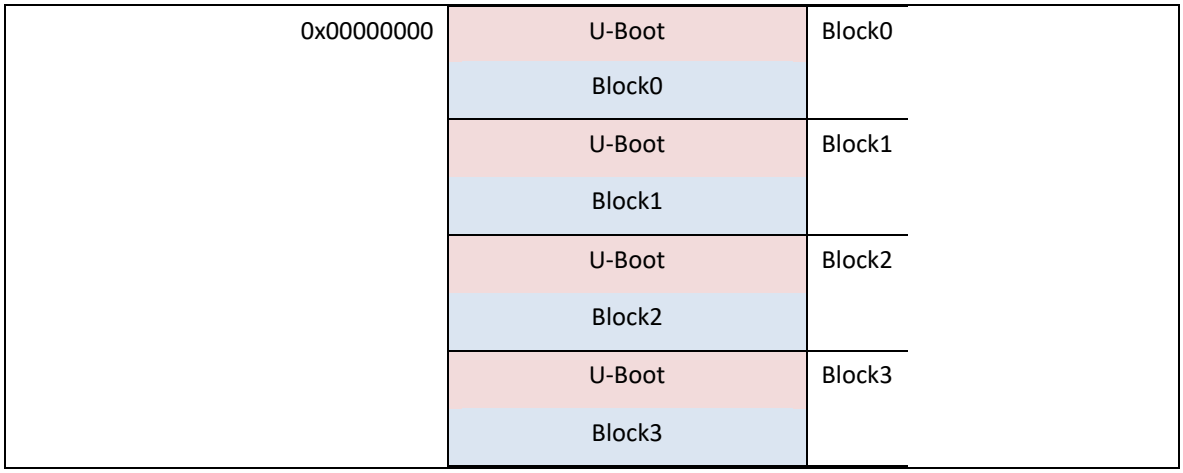


Figure 4-10 NAND Flash Block 0~3

Figure 4-11為實際從NAND Flash 讀取出來的資料，定義說明如下：

Boot Code Marker = {0x20,'T','V','N'}。

Execute Address = 0x00000200。

Image Size = 0x00003F3C。

Reserved = 0xFFFFFFFF。

Page Size = 0x800。

Spare Area = 0x40。

Quad Read cmd = 0xFF。

Read Status cmd = 0xFF。

Write Status cmd = 0xFF。

Status Value = 0xFF。

Dummy Byte = 0xFF。

Reserved = 0xFF。

Reserved = 0xFF。

Reserved = 0xFF。

Reserved = 0xFFFFFFFF。

DDR - Initial Marker = 0x55AA55AA。

DDR - Counter = 0x0000002E。

DDR - Address 0 = 0xB0000264。 DDR - Value0 = 0xC0000018。

以此類推 ・・・

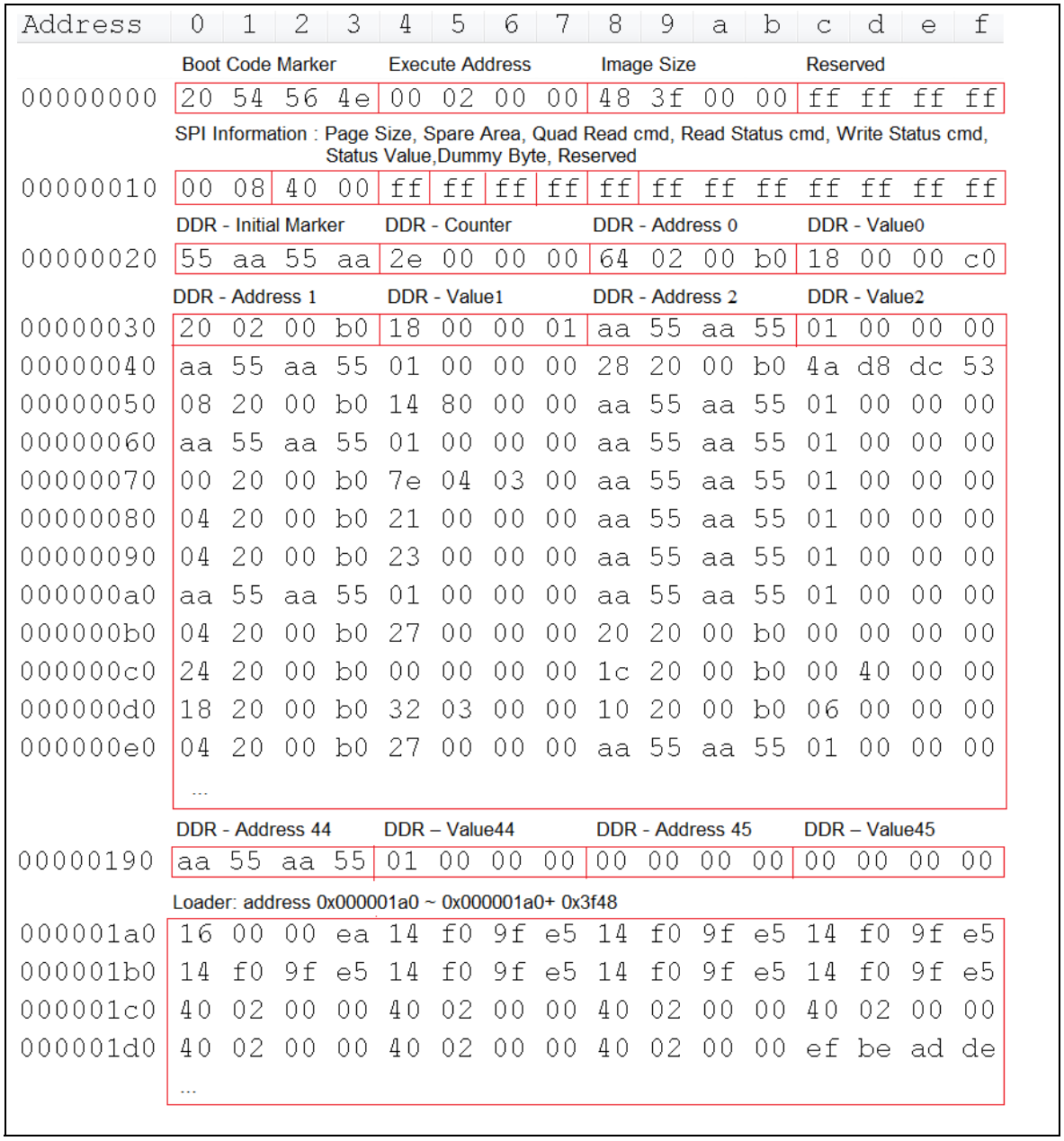
DDR - Address 45 = 0x55AA55AA。DDR – Value45 = 0x1。

DDR – Dummy = 0x00000000。 DDR – Dummy = 0x00000000。

Loader = address 0x00001a0 ~ 0x00001a0 + 0x00003F48。

但是在NAND Flash中Loader 型態有限制如下：

Loader Image + Header + DDR parmater ≤ Block Size



Address	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
	Boot Code Marker				Execute Address				Image Size				Reserved			
00000000	20	54	56	4e	00	02	00	00	48	3f	00	00	ff	ff	ff	ff
	SPI Information : Page Size, Spare Area, Quad Read cmd, Read Status cmd, Write Status cmd, Status Value,Dummy Byte, Reserved															
00000010	00	08	40	00	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff
	DDR - Initial Marker				DDR - Counter				DDR - Address 0				DDR - Value0			
00000020	55	aa	55	aa	2e	00	00	00	64	02	00	b0	18	00	00	c0
	DDR - Address 1				DDR - Value1				DDR - Address 2				DDR - Value2			
00000030	20	02	00	b0	18	00	00	01	aa	55	aa	55	01	00	00	00
00000040	aa	55	aa	55	01	00	00	00	28	20	00	b0	4a	d8	dc	53
00000050	08	20	00	b0	14	80	00	00	aa	55	aa	55	01	00	00	00
00000060	aa	55	aa	55	01	00	00	00	aa	55	aa	55	01	00	00	00
00000070	00	20	00	b0	7e	04	03	00	aa	55	aa	55	01	00	00	00
00000080	04	20	00	b0	21	00	00	00	aa	55	aa	55	01	00	00	00
00000090	04	20	00	b0	23	00	00	00	aa	55	aa	55	01	00	00	00
000000a0	aa	55	aa	55	01	00	00	00	aa	55	aa	55	01	00	00	00
000000b0	04	20	00	b0	27	00	00	00	20	20	00	b0	00	00	00	00
000000c0	24	20	00	b0	00	00	00	00	1c	20	00	b0	00	40	00	00
000000d0	18	20	00	b0	32	03	00	00	10	20	00	b0	06	00	00	00
000000e0	04	20	00	b0	27	00	00	00	aa	55	aa	55	01	00	00	00
	...															
	DDR - Address 44				DDR – Value44				DDR - Address 45				DDR – Value45			
00000190	aa	55	aa	55	01	00	00	00	00	00	00	00	00	00	00	00
	Loader: address 0x000001a0 ~ 0x000001a0+ 0x3f48															
000001a0	16	00	00	ea	14	f0	9f	e5	14	f0	9f	e5	14	f0	9f	e5
000001b0	14	f0	9f	e5	14	f0	9f	e5	14	f0	9f	e5	14	f0	9f	e5
000001c0	40	02	00	00	40	02	00	00	40	02	00	00	40	02	00	00
000001d0	40	02	00	00	40	02	00	00	40	02	00	00	ef	be	ad	de
	...															

Figure 4-11 Address 0x00000000~0x00001a0 of NAND Flash

4.3.1.2 Data型態

主要將指定的檔案放入到NAND Flash中所指定的位址，不做任何其他的處理。依據輸入的Image start offset的值(需要對齊Page Size，Page Size是依據NAND Flash規格所決定)，決定將Data存放在NAND Flash的哪個位址，如果Image start offset = 0x10000，則將Data 存放到 NAND Flash 0x10000 的位址，此方式主要可以幫助使用者依據個人需求配置NAND Flash，例如當Linux Kernel 需要透過Loader型態的Image帶起來，Image設定帶起來的位址為0x8000，則需要將Linux Kernel當作Data型態，配置到0x8000的位址。

Nuwriter Tool支援YAFFS2與UBIFS兩種檔案系統的Image以Data型態燒錄至NAND Flash對應的位址，讓使用者可以透過uBoot或Linux來讀取檔案系統。

YAFFS2製作In-band tags Image命令如下：(yaffs2的tag儲存在DATA區塊中)

```
# mkyaffs2 --inband-tags -p 2048 rootfs rootfs_yaffs2.img
```

--inband-tags：yaffs2的tag儲存在DATA區塊。

-p：設定NAND Flash頁的大小(Page Size)。

即可將rootfs資料夾壓縮成*rootfs_yaffs2.img*，再透過NuWriter以Data型態燒錄至相對應NAND Flash的位址。輸入下列命令即可將YAFFS2 inband-tags 檔案系統掛在flash資料夾中：

```
mount -t yaffs2 –o “inband-tags” /dev/mtdblock2 /flash
```

YAFFS2的指令可以在yaffs2utils套件中找到。

UBIFS製作Image命令如下：

```
# mkfs.ubifs -F -x lzo -m 2048 -e 126976 -c 732 -o rootfs_ubifs.img -
d ./rootfs
# ubinize -o ubi.img -m 2048 -p 131072 –O 2048 -s 2048 rootfs_ubinize.cfg
```

mkfs.ubifs 使用的參數說明如下：

-F ：設定檔案系統未使用的空間優先mount.

-x： 壓縮的格式，"lzo", "favor_lzo", "zlib" 或 "none" (預設："lzo")

-m：最小的I/O操作的大小，也就是NAND Flash一個頁的大小。

-e：邏輯擦除塊的大小(logical erase block size)。因為實體擦除塊(PEB)為128KiB，所以邏輯擦除塊設定為124KiB=126976。

-c：最大的擦除塊的號碼(maximum logical erase block count)。

-o：輸出檔案。

ubinize使用的參數說明名如下：

-o：輸出檔案。

-m：最小輸入/輸出的大小，也就是NAND Flash一個頁的大小。

-p：實體擦除塊大小，128KiB=131072。

-O：VID檔頭位移位置。

-s：使用最小輸入/輸出的大小，存放UBI檔頭。

rootfs_ubinize.cfg 內容如下：

```
[rootfs-volume]
mode=ubi
image=rootfs_ubifs.img
vol_id=0
vol_size=92946432
vol_type=dynamic
vol_name=system
vol_flags=autoresize
```

即可將rootfs資料夾壓縮成*ubi.img*，再透過NuWriter以Data型態燒錄至相對應NAND Flash的位址。

輸入下列命令即可將UBIFS檔案系統掛在flash資料夾中：

需要參考/sys/class/misc/ubi_ctrl/dev內容，假設內容為 10：56，則設定如下：

```
mknod /dev/ubi_ctrl c 10 56
ubiattach /dev/ubi_ctrl -p /dev/mtd2
mount -t ubifs ubi0:system /flash
```

UBIFS相關指令可以在mtd-utils套件中找到。

Linux內核也必須將對應的檔案系統設置後才可正常運作設置方式如下：

YAFFS2：

```
File systems  --->
      [*] Miscellaneous filesystems  --->
            <*>   yaffs2 file system support
            <*>   Autoselect yaffs2 format
            <*>   Enable yaffs2 xattr support
```

UBIFS：

```
Device Drivers  --->
      -*- Memory Technology Device (MTD) support  --->
             <*>   Enable UBI - Unsorted block images  --->
File systems  --->
      [*] Miscellaneous filesystems  --->
            <*>   UBIFS file system support
            [*]     Advanced compression options
            [*]       LZO compression support
            [*]       ZLIB compression support
```

相關套件編譯：

目錄	描述

lzo-2.09.tar.gz	壓縮/解壓縮工具.
	交叉編譯命令如下:
	$ cd lizo-2.09
	$ ./configure --host=arm-linux --prefix=$PWD/../install
	$ make
	$ make install

libuuid-1.0.3.tar.gz	產生獨立的序號工具. 交叉編譯命令如下:
	$ cd libuuid-1.0.3
	$ ./configure --host=arm-linux --prefix=$PWD/../install
	$ make
	$ make install

mtd-utils.tar.gz	mtd-utils源碼. 交叉編譯命令如下:
	需要使用到lzo-2.09.tar.gz套件和libuuid-1.0.3.tar.gz套件
	$ cd mtd-utils
	$ export CROSS=arm-linux-
	$ export WITHOUT_XATTR=1
	$ export DESTDR=$PWD/../install
	$ export LZOCPPFLAGS=-I/home/install/include
	$ export LZOLDFLAGS=-L/home/install/lib
	$ make
	$ make install

yaffs2utils.tar.gz	yaffs2命令工具 $ make

4.3.1.3 Environment型態

主要設定Loader 的環境變數，檔案放入到NAND Flash中所指定的位址，讓Loader 讀取時做相對應的動作。依據輸入的Image start offset的值(需要對齊Page Size)，決定將Enviroment存放在NAND Flash的哪個位址，如果Image start offset = 0x10000，則將Enviroment存放到 NAND Flash 0x10000 的位址。Loader環境變數的默認位移設置是 0x80000，如果使用者在這裡做不同的設置，也請同時修改 U-Boot 的設置。

env.txt 存放的是 U-Boot 的環境變數及其數值，內容舉例如下:

```
baudrate=115200
bootdelay=3
ethact=emac
ethaddr=00:00:00:11:66:88
stderr=serial
stdin=serial
stdout=serial
```

4.3.1.4 Pack型態

依據輸入Pack.bin的內容，決定將Pack內容中的Image放到相對應NAND Flash的位址。Pack.bin的原理可以參考4.7章節。

4.3.1.5 壞塊處理

NAND Flash支援四種型態的方式燒入到NAND Flash中，對於壞塊的寫入處理直接跳過。假設有一資料將寫到NAND Flash的Block 0, Block 1, Block 2, Block 3和Block 4，但是當中的Block 3偵測為壞塊時，會跳過Block 3的寫入處理，往下一個Block繼續寫入。也就是說，寫入的Block變成Block 0, Block 1, Block 2, Block 4和Block 5。

4.3.2 操作方法

4.3.2.1 新增Image

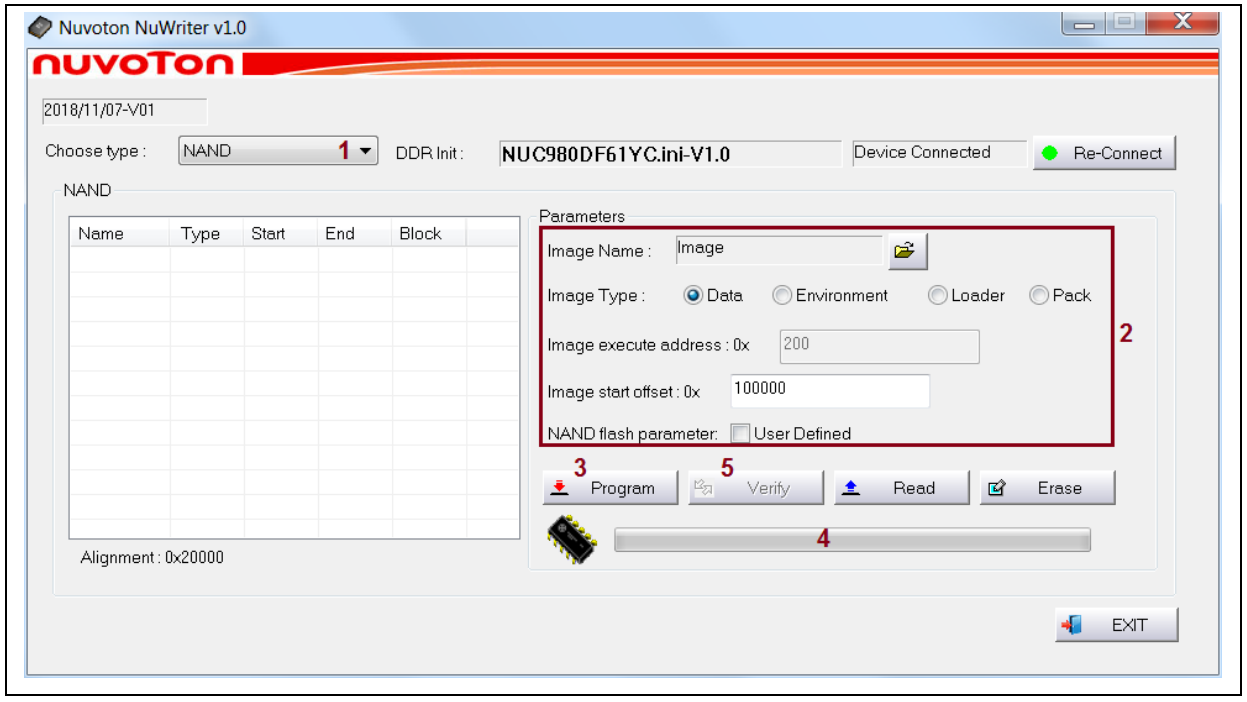


Figure 4-12 NAND – New Image

依照Figure 4-12步驟即可以完成新增Image檔案，操作步驟如下：

1. 選擇“**NAND**”模式， 表格只會紀錄當次燒錄的Image檔案，並不會讀取NAND Flash中Image的資料。
2. 輸入Image檔案資料：
   - Image Name 選擇要燒錄的 Image檔案。
   - Image Type 選擇Image型態。
   - Image execute address 設置Image執行位置, 依編譯設定而輸入，只有在Loader 型態才有效。
   - Image start offset 設置Image燒錄在NAND Flash的位址。
   - NAND flash parameter 勾選 “**User Defined**”會跳出視窗，提供NAND參數設定。
3. 按下”**Program**”。
4. 等待進度表完成，表格將會顯示這次燒錄完成的Image檔案。
5. 燒錄完成後，可以選擇按下“**Verify**”即可確認燒入資料是否正確。

4.3.2.2 *自訂NAND參數*

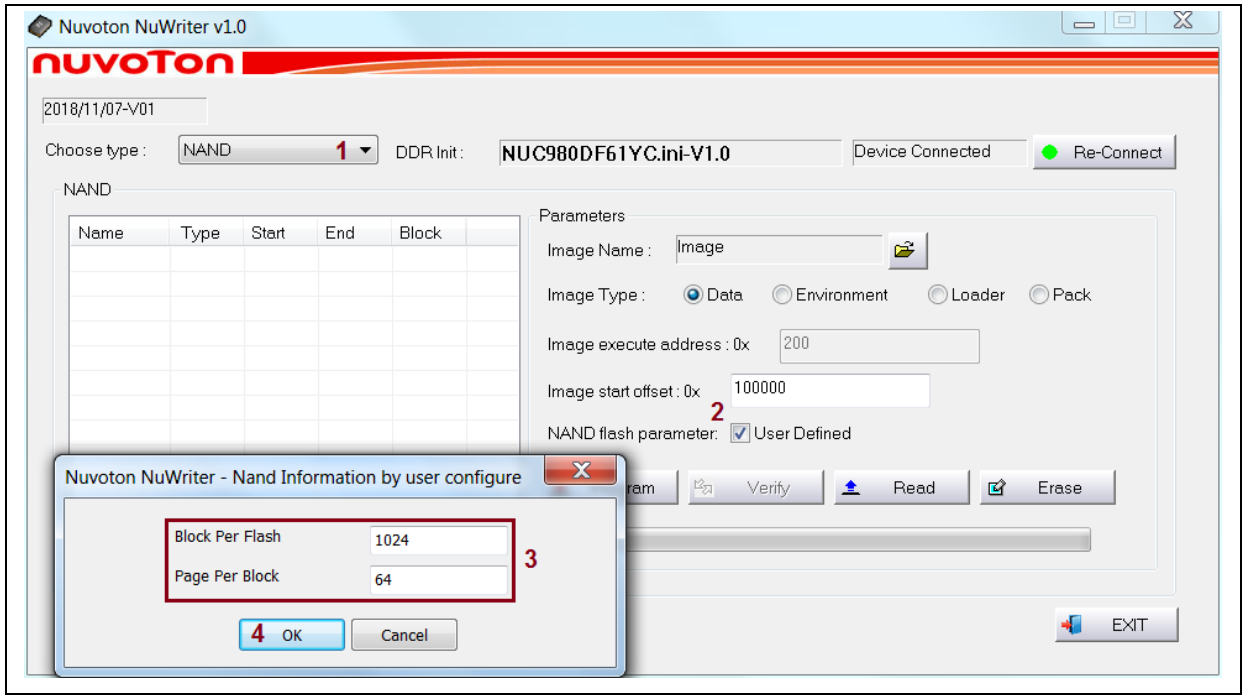


Figure 4-13 NAND – Configure Parameters

NuWriter會自動偵測NAND的ID來決定它的參數，使用者也可以自己依據NAND Flash規格來設定NAND的參數。依照Figure 4-13步驟即可設定NAND參數：

1. 選擇 “**NAND**” 模式。
2. 勾選 “**User Defined**”會跳出視窗，提供NAND參數設定。
3. 輸入Block Per Flash(總共多少個Block)及Page Per Block(每個Block有幾個Page) 。
4. 按下 “**OK**”，即可完成參數設定。

4.3.2.3 *讀取Image*

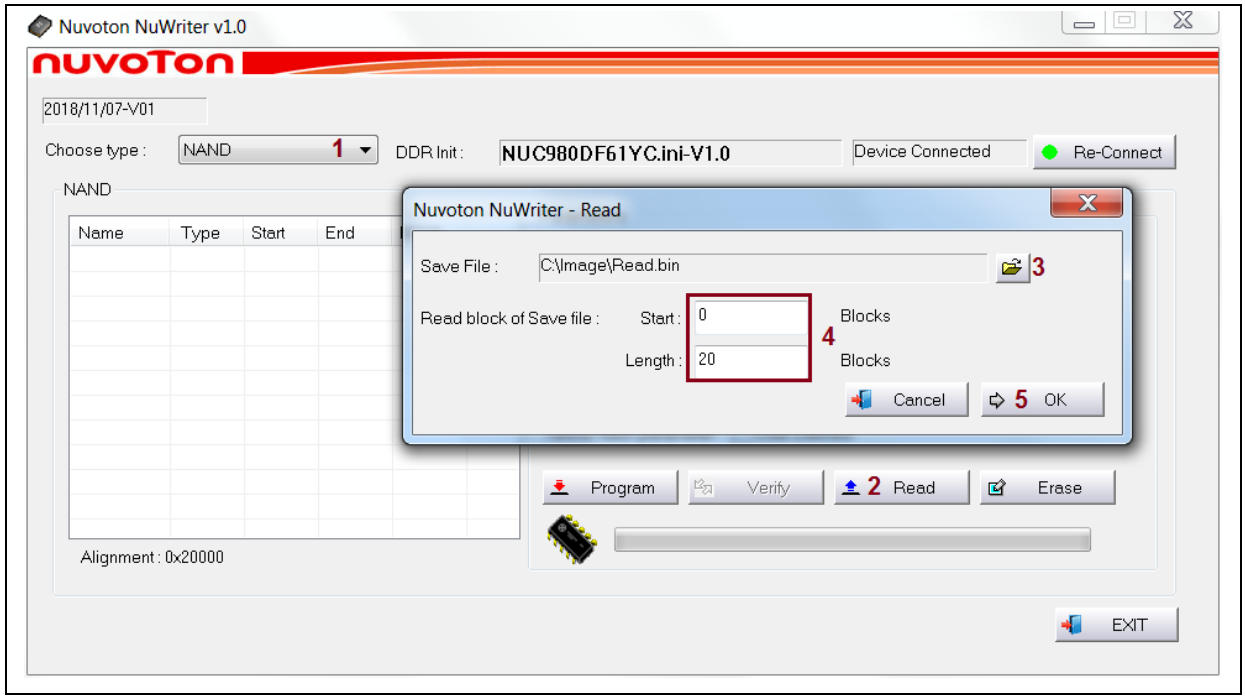


Figure 4-14 NAND – Read

依照Figure 4-14步驟即可以完成讀取Image：

1. 選擇 “**NAND**” 模式。
2. 按下 “**Read**”。
3. 選擇要儲存檔案的位置。
4. 輸入欲讀取的Block起始位置與Block長度(每一Block大小依據NAND Flash規格來決定) 。
5. 按下“**OK**”，即可完成Image讀取。

4.3.2.3.1 *讀取模式*

Nand Flash 包含 OOB(out of band) 區域，NuWriter 可以透過設定 *NuWriter/path.ini* 中的 ChipReadWithBad選項來選取欲讀取區域。

1. ChipReadWithBad=0時為預設值，讀取時只會讀回資料的部分。
2. ChipReadWithBad=1時，讀取時會讀取資料加上OOB區域。

```
[SDRAM]
…

[TARGET]
CHIP=0
DDR=0
DDRADDRESS=16
TimeEnable=1
Timeus=5000
ChipEraseWithBad=0
ChipReadWithBad=0
```

```
ChipWriteWithOOB=0
FirstDelay=500
…
```

4.3.2.4 *移除 Image*

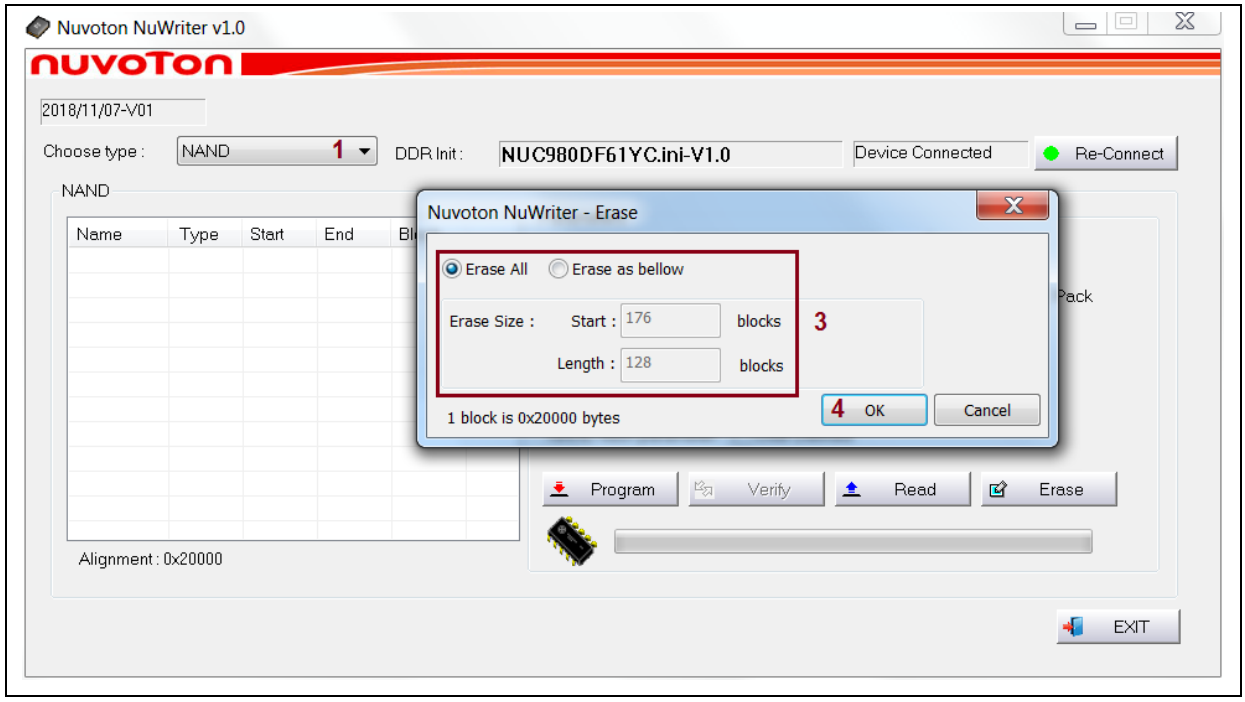


Figure 4-15 NAND – Erase

依照Figure 4-15步驟即可以完成移除Image檔案：

1. 選擇 “**NAND**” 模式 。
2. 按下 “**Erase**”
3. 選擇 “**Erase all**” 或選擇 “**Erase as below**”
   - 輸入擦除的Blocks(每一Block大小依據NAND Flash規格來決定)
4. 按下 “**OK**”，即可完成擦除。

4.3.2.4.1 *移除模式*

Nand Flash 包含 OOB(out of band) 區域，NuWriter 可以透過設定 NuWriter/path.ini 中的 ChipReadWithBad選項來選取欲移除區域。

1. ChipEraseWithBad=0時為預設值，清除時只會清除資料。
2. ChipEraseWithBad=1時，清除時會將資料與OOB區域都會清除。

```
[SDRAM]
…

[DEFAULT]
PAGE_IDX=3
```

```
[TARGET]
CHIP=0
DDR=0
DDRADDRESS=16
TimeEnable=1
Timeus=5000
ChipEraseWithBad=0
ChipReadWithBad=0
ChipWriteWithOOB=0
FirstDelay=500
…
```

4.4 SPI 模式

4.4.1 Image檔案型態

4.4.1.1 Loader 型態

主要是作為開機的第一支程式，當NUC980系列晶片上電後會讀取Loader 型態中的Image並執行，不限定一定要存放Loader ，一般程式也是可以使用Loader 型態來作為上電後第一支執行程式。Loader 增加32bytes的檔頭後和DDR參數合併，Figure 4-16為Loader型態的Image格式。

	0x0		0x4				0x8				0xC
0x00	Boot Code Marker		Execute Address				Image Size				Reserved
0x10	Page Size	Spare Area	Quad Read cmd	Read Status cmd	Write Status cmd	Status Value	Dummy Byte	Reserved	Reserved	Reserved	Reserved
0x20	DDR - Initial Marker		DDR - Counter				DDR - Address 0				DDR – Value0
0x30	DDR - Address 1		DDR - Value 1				DDR - Address 2				DDR - Value 2
0x40	DDR - Address 3		DDR - Value 3				DDR - Address 4				DDR - Value 5
0x50	DDR - Address 5		DDR - Value 5				DDR - Dummy				DDR - Dummy
0x60	uBoot										

Figure 4-16 Boot Code Header

詳細的內容可以參考NAND Flash模式4.3.1.1章節。檔案合併以後再燒入到SPI Flash中0x0的位址，Figure 4-17為示意圖：

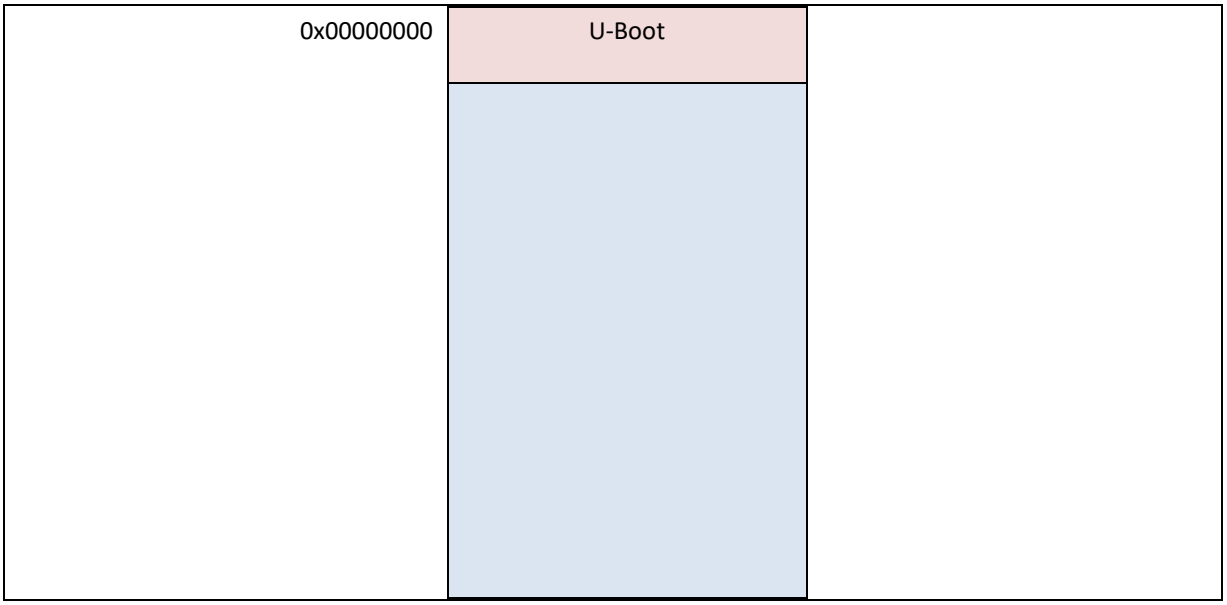


Figure 4-17 SPI Flash

4.4.1.2 Data 型態

主要將指定的檔案放入到SPI Flash中所指定的位址，不做任何其他的處理。依據輸入的Image start offset的值(需要對齊Block Size，Block Size是依據SPI Flash規格所決定)，決定將Data存放在SPI Flash的哪個位址，如果Image start offset = 0x10000，則將Data 存放到 SPI Flash 0x10000 的位址，此方式主要可以幫助使用者依據個人需求配置SPI Flash，例如當Linux Kernel需要由Loader 型態中Image帶起來時，Linux Kernel帶起來的位址為0x8000，則需要將Linux Kernel當作Data型態，配置到0x8000的位址。

4.4.1.3 Environment 型態

主要設定Loader 的環境變數，檔案放入到SPI Flash中所指定的位址，讓Loader 讀取時做相對應的動作。依據輸入的Image start offset的值(需要對齊Block size)，決定將Enviroment存放在SPI Flash的哪個位址，如果Image start offset = 0x10000，則將Enviroment存放到SPI Flash0x10000 的位址。

4.4.1.4 Pack 型態

依據輸入Pack.bin 的內容，決定將Pack內容中的Image 放到相對應SPI Flash的位址。Pack.bin的原理可以參考Pack模式4.7章節。

4.4.2 操作方法

4.4.2.1 新增Image

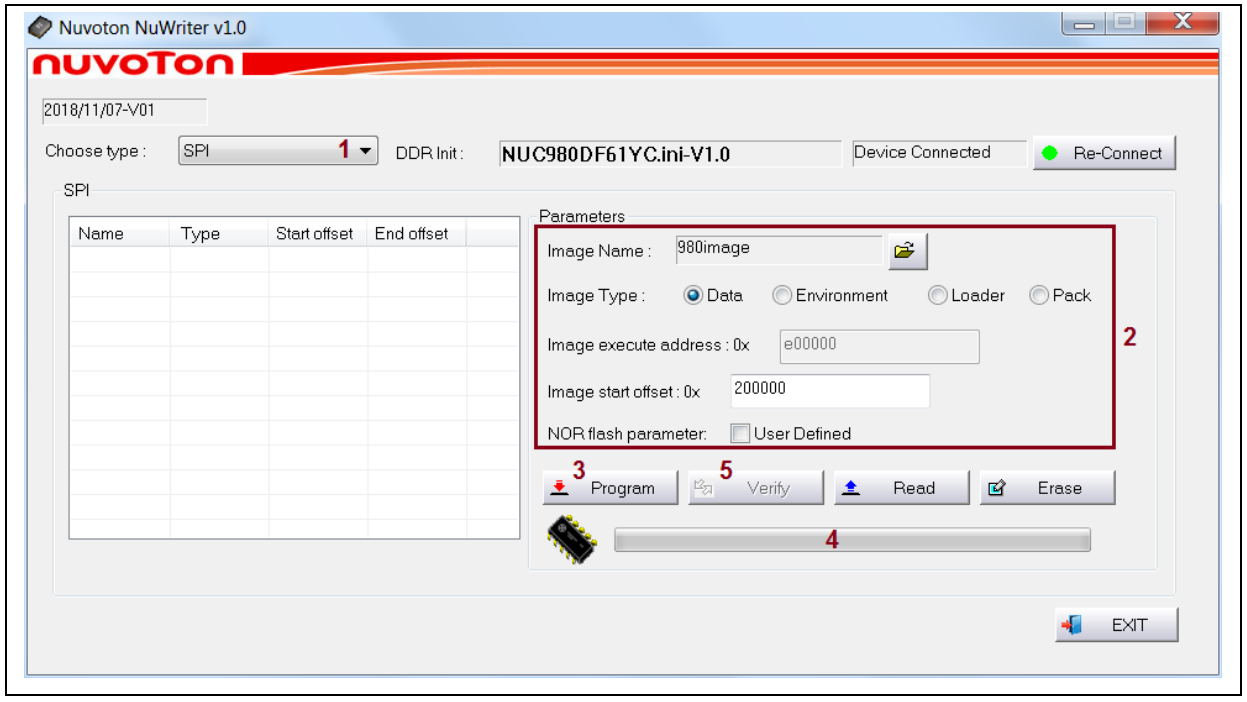


Figure 4-18 SPI – New Image

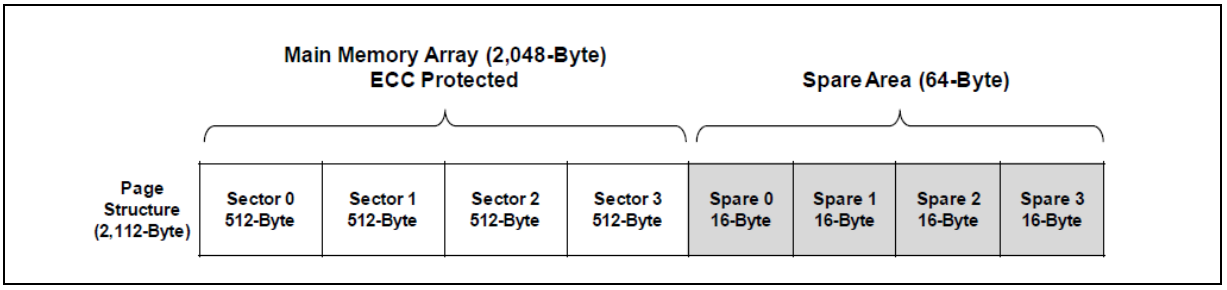
依照Figure 4-18步驟即可以完成新增Image檔案：

1. 選擇 “**SPI**” 模式，表格只會紀錄當次燒錄的Image檔案，並不會讀取SPI Flash中Image的資料。
2. 輸入Image檔案資料：
   - Image Name 選擇要燒錄的 Image檔案
   - Image Type 選擇Image型態
   - Image execute address 設置Image執行位置，依編譯設定而輸入，只有在Loader 型態才有效
   - Image start offset設置Image燒錄在SPI Flash的位址
   - NOR flash parameter 勾選 “**User Defined**”會跳出視窗，提供SPI NOR Flash參數設定。
3. 按下 “**Program**”。
4. 等待進度表完成，表格將會顯示這次燒錄完成的Image檔案。
5. 燒錄完成後，可選擇按下“**Verify**”即可確認燒入資料是否正確。

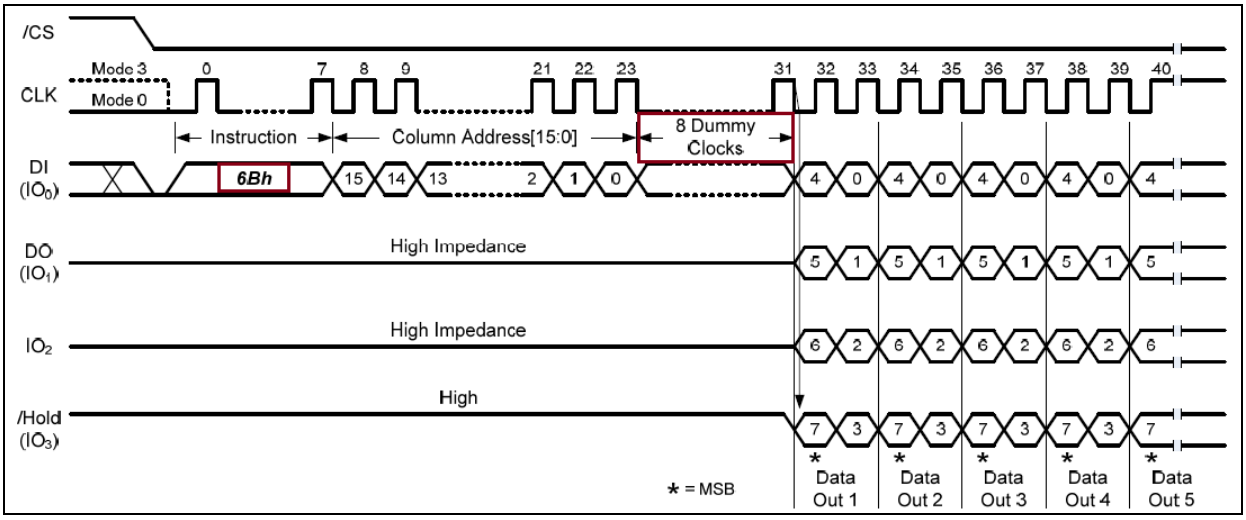
4.4.2.2 自訂SPI參數

NuWriter會自動偵測SPI Flash的ID來決定它的參數，使用者也可以自己依據SPI Flash規格透過NuWriter設定參數。使用者可以自訂SPI的Page Size、Spare Area大小；此外，也可以設定SPI Quad模式讀取命令、讀取 SPI Flash狀態寄存器命令、寫入 SPI Flash狀態寄存器命令、Quad 模式數值，以及 Dummy Byte個數。

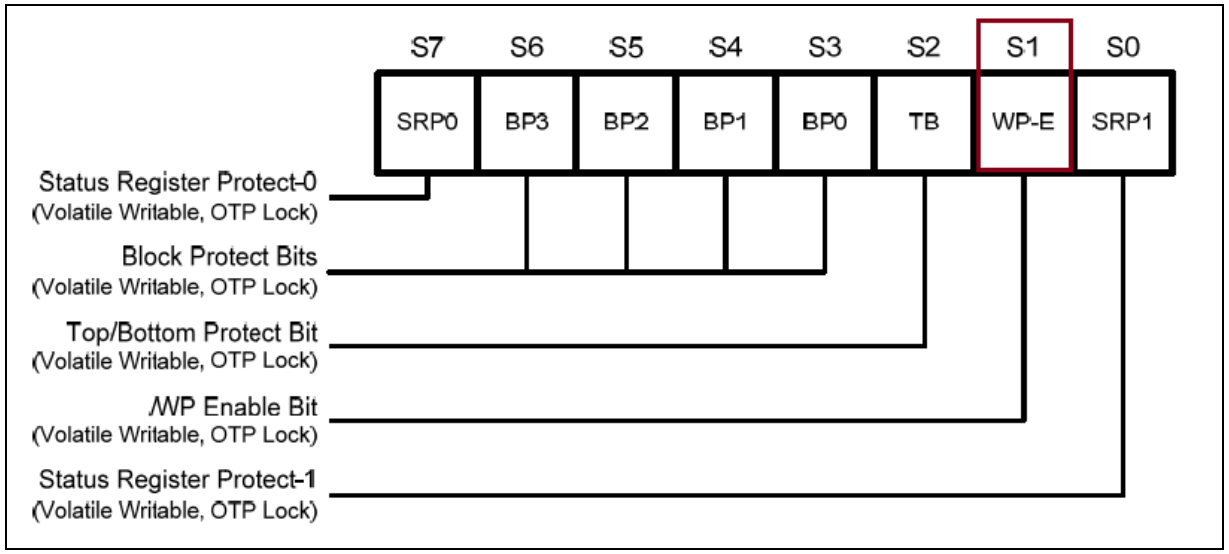
SPI Flash的Page Size及Spare Area大小會隨著不同的 SPI Flash型號而不同，相關的資料可以從SPI Flash規格書中獲得。以下為 SPI Flash規格書中描述 Page 大小以及 Spare Area 大小的範例。



Quad 模式格式為: QuadReadCmd=Quad模式命令，ReadStatusCmd=讀取狀態寄存器命令，WriteStatusCmd=寫入狀態寄存器命令，StatusValue=狀態寄存器中Quad模式狀態所在的位置，DummyByte= Quad 命令所需的 Dummy Byte個數，命令數值是採用 16 進制。SPI Quad 模式讀取命令同樣也可以從規格書中取得。



SPI Quad模式需要把狀態寄存器中的WP-E位元設置為1，此值對應Nuwriter的StatusValue，狀態寄存器的位元的定義需參考SPI Flash規格書，下圖是從某SPI Flash規格書裡所擷取。



依照Figure 4-19步驟即可以完成設定SPI Flash參數：

1. 選擇 “**SPI**” 模式。
2. 勾選 “**User Defined**”會跳出視窗，提供SPI Flash參數設定。

3. 根據SPI Flash規格輸入Quad Read Command、Read Status Command、Write Status Command、Status Value及Dummy Byte。
4. 按下 “**OK**”， 即可完成參數設定。

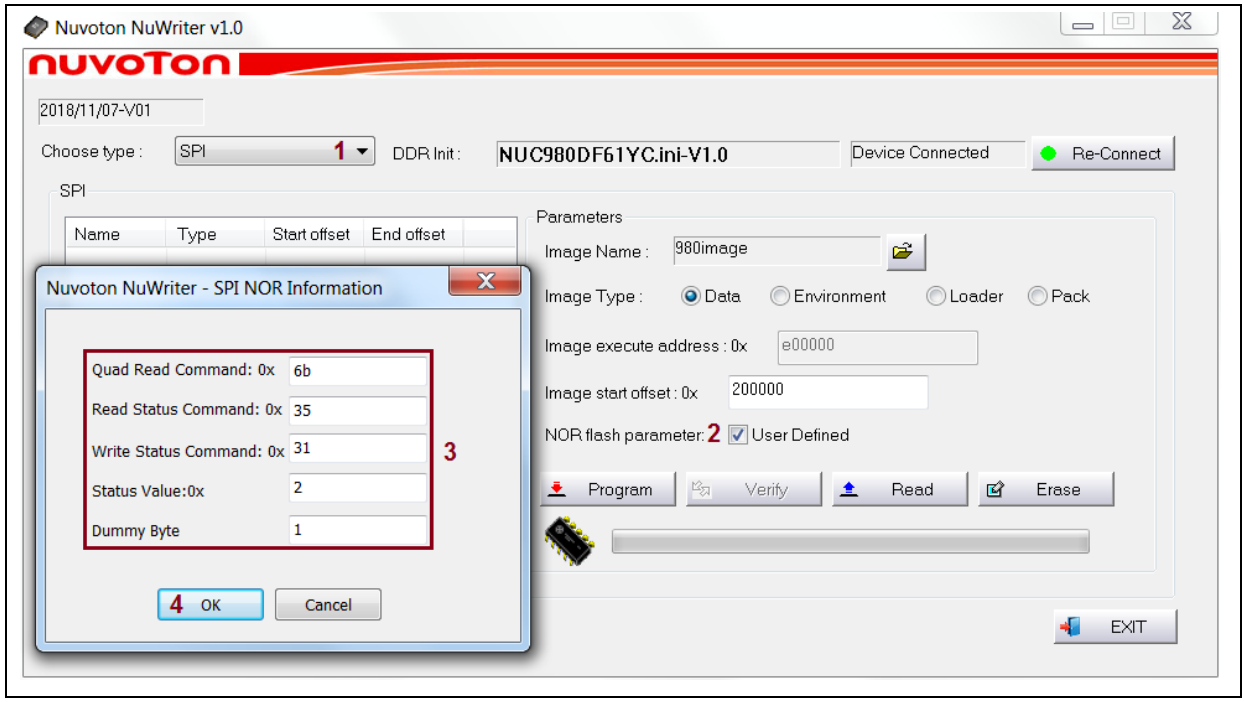


Figure 4-19 SPI – Configure Parameters

4.4.2.3 讀取Image

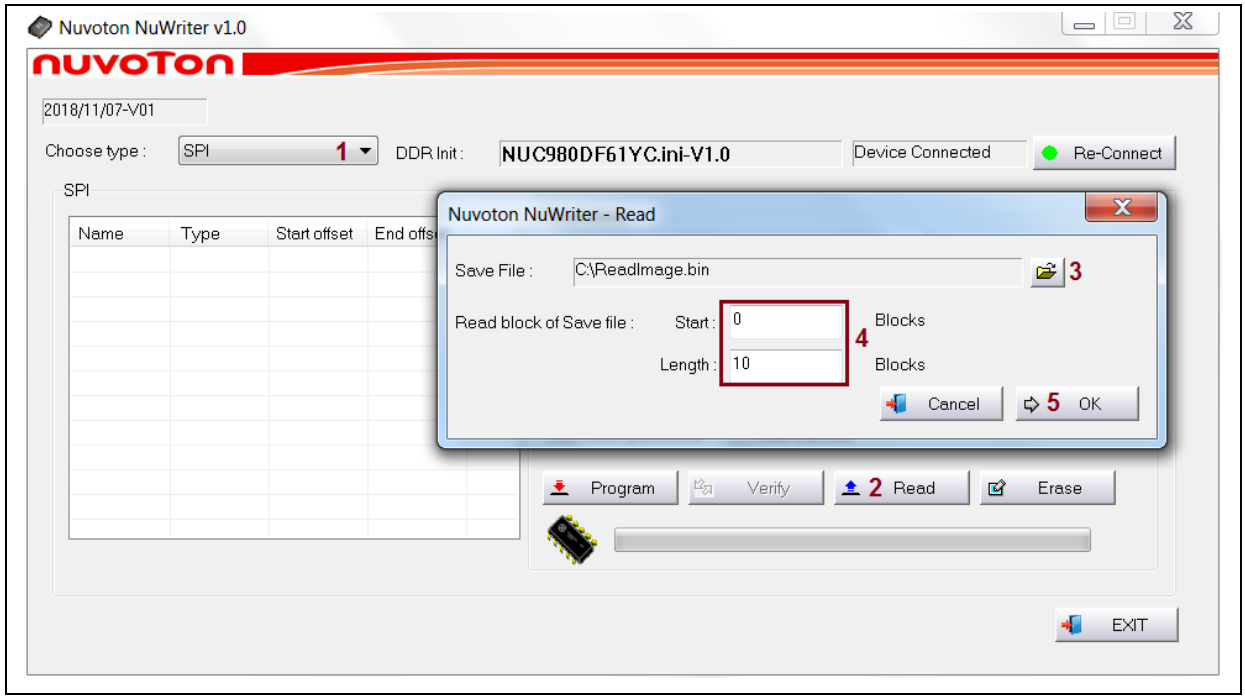


Figure 4-20 SPI – Read Image

依照Figure 4-20步驟即可以完成讀取Image：

1. 選擇 “**SPI**”模式。
2. 按下 “**Read**”。
3. 選擇要儲存檔案的位置。
4. 輸入欲讀取的Block起始位置與Block長度(每一Block大小依據SPI Flash規格來決定) 。
5. 按下 “**OK**”， 即可完成Image讀取。

4.4.2.4 移除 Image

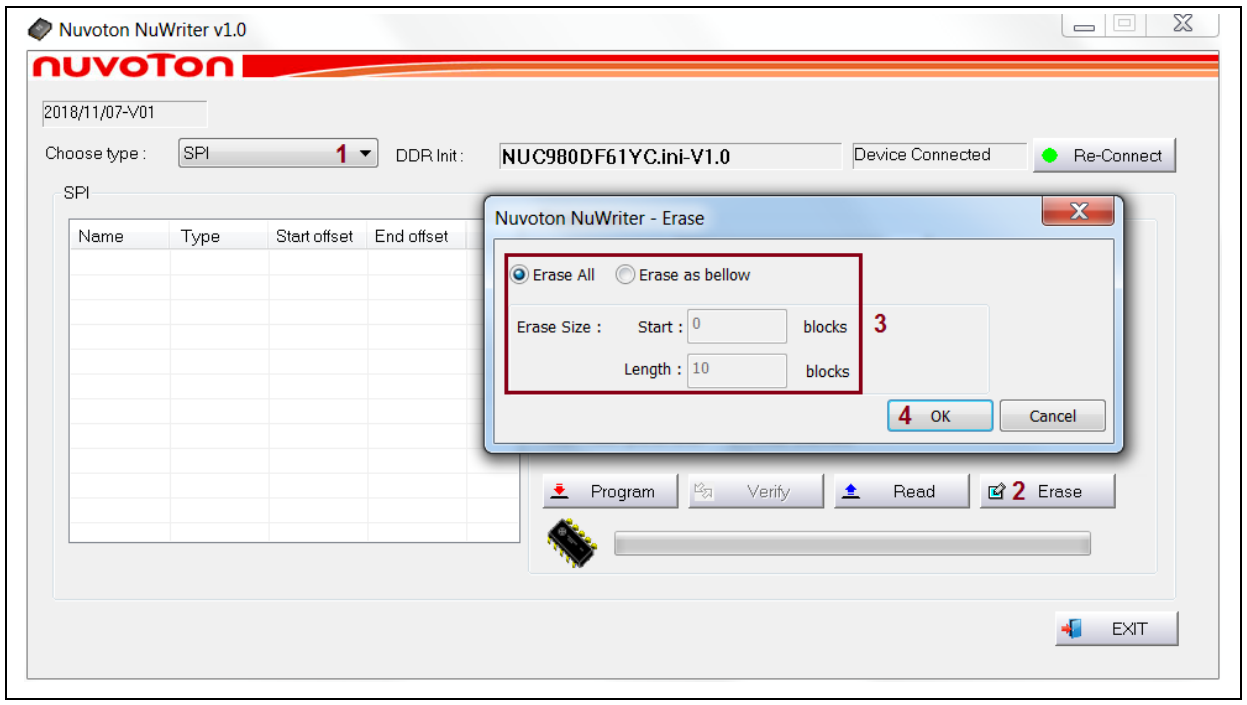


Figure 4-21 SPI – Erase

依照Figure 4-21步驟即可以完成移除Image檔案：

1. 選擇 “**SPI**” 模式 。
2. 按下 “**Erase**”
3. 選擇 “**Erase all**” 或選擇 “**Erase as below**”
   - 輸入擦除的Blocks(每一Block大小依據SPI Flash規格來決定)
4. 按下 “**OK**”，即可完成擦除。

4.5 eMMC/SD 模式

開發板上有兩組eMMC/SD，分別為eMMC0/SD0及eMMC1/SD1，使用者可以透過Power-on Setting的指撥開關SW2.9和SW2.10去選擇使用哪一組eMMC/SD。開發板上幾乎放置了NUC980系列晶片的所有功能，因此使用上需要注意某些功能的腳位會共用。eMMC0/SD0在開發板上跟NAND Flash共用腳位，所以使用eMMC/SD0模式時要把指撥開關SW3和SW4設定為Off。

4.5.1 Image檔案型態

4.5.1.1 Loader 型態

主要是作為開機的第一支程式，當NUC980系列晶片上電後會讀取Loader 型態中的Image並執行，不限定一定要存放Loader ，一般程式也是可以使用Loader 型態來作為上電後第一支執行程式。Loader 增加32bytes的檔頭後和DDR參數合併，方式如下：

	0x0		0x4				0x8				0xC
0x00	Boot Code Marker		Execute Address				Image Size				Reserved
0x10	Page Size	Spare Area	Quad Read cmd	Read Status cmd	Write Status cmd	Status Value	Dummy Byte	Reserved	Reserved	Reserved	Reserved
0x20	DDR - Initial Marker		DDR - Counter				DDR - Address 0				DDR – Value0
0x30	DDR - Address 1		DDR - Value 1				DDR - Address 2				DDR - Value 2
0x40	DDR - Address 3		DDR - Value 3				DDR - Address 4				DDR - Value 5
0x50	DDR - Address 5		DDR - Value 5				DDR - Dummy				DDR - Dummy
0x60	uBoot										

Figure 4-22 Boot Code Header

詳細的內容可以參考NAND Flash模式4.3.1.1章節。Boot code header 與uBoot image合併以後再燒入到eMMC/SD中0x400的位址，Figure 4-23為示意圖：

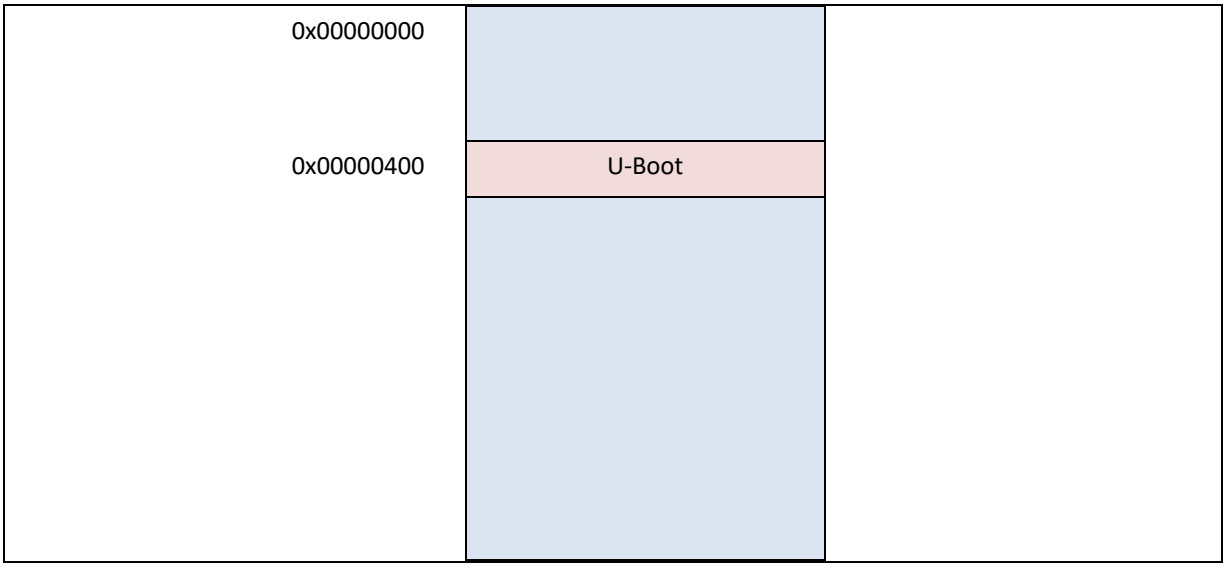


Figure 4-23 eMMC/SD

4.5.1.2 Data 型態

主要將指定的檔案放入到eMMC/SD中所指定的位址，不做任何其他的處理。依據輸入的Image start offset的值(需要對齊512bytes)，決定將Data存放在eMMC/SD的哪個位址，如果Image start offset =

0x10000，則將Data存放到eMMC/SD 0x10000的位址，此方式主要可以幫助使用者依據個人需求配置eMMC/SD，例如當Linux Kernel 需要由Loader 型態中Image帶起來時，Linux Kernel帶起來的位址為0x8000，則需要將Linux Kernel當作Data型態，配置到0x8000的位址。

4.5.1.3 Environment 型態

主要設定uBoot的環境變數，檔案放入到eMMC中所指定的位址，讓Loader 讀取時做相對應的動作。依據輸入的Image start offset的值(需要對齊512bytes)，決定將Enviroment存放在eMMC/SD的哪個位址，如果Image start offset = 0x10000，則將Enviroment存放到 eMMC 0x10000 的位址。

4.5.1.4 Pack 型態

依據輸入Pack.bin的內容，決定將Pack內容中的Image 放到相對應eMMC/SD的位址。Pack.bin的原理可以參考Pack模式章節。

4.5.1.5 格式化(FAT32)

因為部分空間需要存放Loader型態的Image或者其他型態的Image，所以必須設定保留空間來存放，不給使用者去做任何的修改，保留空間是以512bytes為單位，使用者可以依據自己的需要來設定保留空間的大小。注意：修改此參數可能造成既有的Image或檔案系統格式損毀。

FAT32檔案系統是支援多個磁碟分割區，但是受限於主開機紀錄中檔案分配表的固定結構，一個硬碟最多只能切出多達4個分割區。

4.5.2 操作方法

4.5.2.1 新增Image

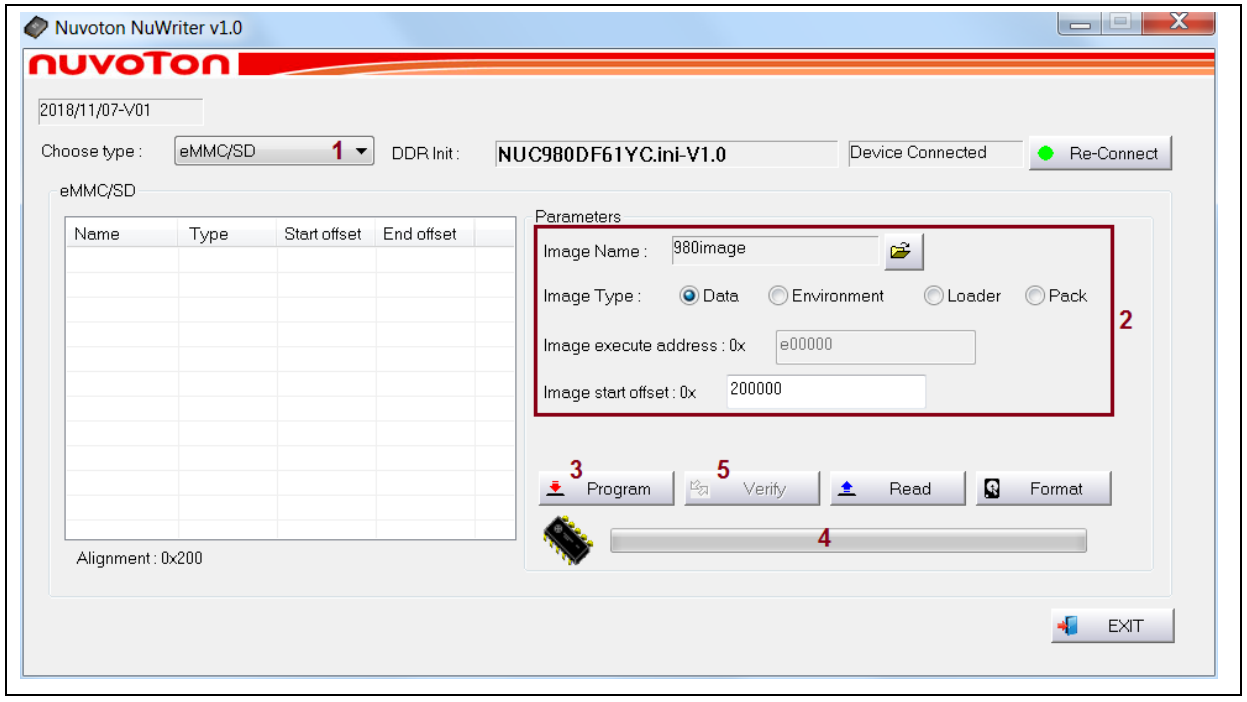


Figure 4-24 eMMC/SD– New Image

依照Figure 4-24步驟即可以完成新增Image檔案：

1. 選擇 “**eMMC/SD**” 模式，表格只會紀錄當次燒錄的Image檔案，並不會讀取eMMC/SD中Image的資料。
2. 輸入Image檔案資料：

- Image Name 選擇要燒錄的Image檔案
- Image Type 選擇Image型態
- Image execute address 設置Image執行位置, 依編譯設定而輸入，只有在Loader型態才有效
- Image start offset設置Image燒錄在eMMC/SD的位址

3. 按下”**Program**”。
4. 等待進度表完成後，表格將會顯示這次燒錄完成的Image檔案。
5. 燒錄完成後，可以選擇按下**“Verify”**即可確認燒入資料是否正確。

4.5.2.2 讀取Image

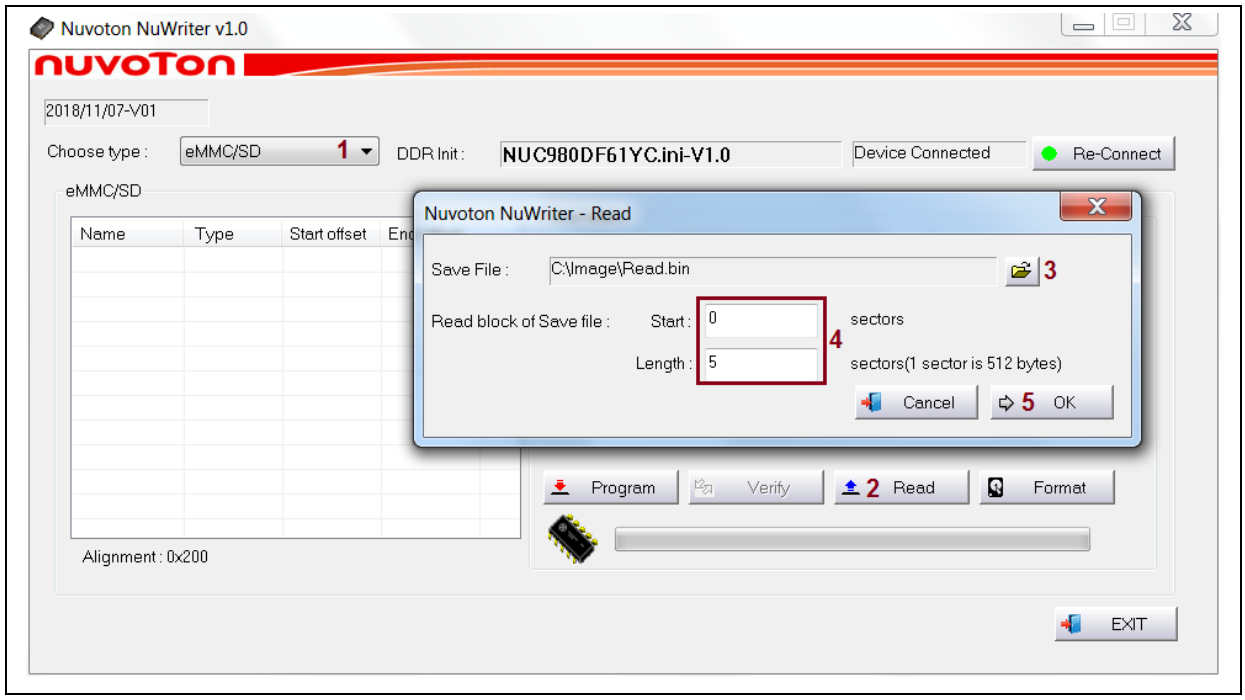


Figure 4-25 eMMC/SD– Read

依照Figure 4-25步驟即可以完成讀取eMMC/SD：

1. 選擇 “**eMMC/SD**” 模式。
2. 按下 “**Read**”。
3. 輸入儲存的檔案。
4. 輸入欲讀取的sector起始位置與sector長度(1 sector is 512 bytes) 。
5. 按下 “**OK**”，即可完成。

4.5.2.3 格式化 (FAT32)

依照Figure 4-26、Figure 4-27和Figure 4-28 eMMC/SD– Format(3)步驟即可以完成eMMC/SD格式化：

1. 選擇 “**eMMC/SD**” 模式。
2. 按下 “**Format**”。
3. 輸入保留空間(單位為512bytes)。注意：修改此參數可能造成既有的Image或檔案系統格式損毀。
4. 按下 “**Set**”，套用保留空間大小。
5. 設定完保留空間後，下方有四個滑動捲軌可以讓使用者自行配置eMMC/SD的空間。使用者操作滑動捲軌來決定分割區大小。
   - 1st PartitionSize: 第一個分割區大小
   - 2nd PartitionSize: 第二個分割區大小

- 3rd PartitionSize: 第三個分割區大小
- 4th PartitionSize: 第四個分割區大小

6. 每次設定一個分割區大小，按下”**Set**”讓分割區大小生效。
7. 按下”**Add**”即可繼續新增下一個分割區大小，反覆5~7最多可以將eMMC/SD的空間分割成四個分割區。
8. 按下 “**OK**”，即可完成。

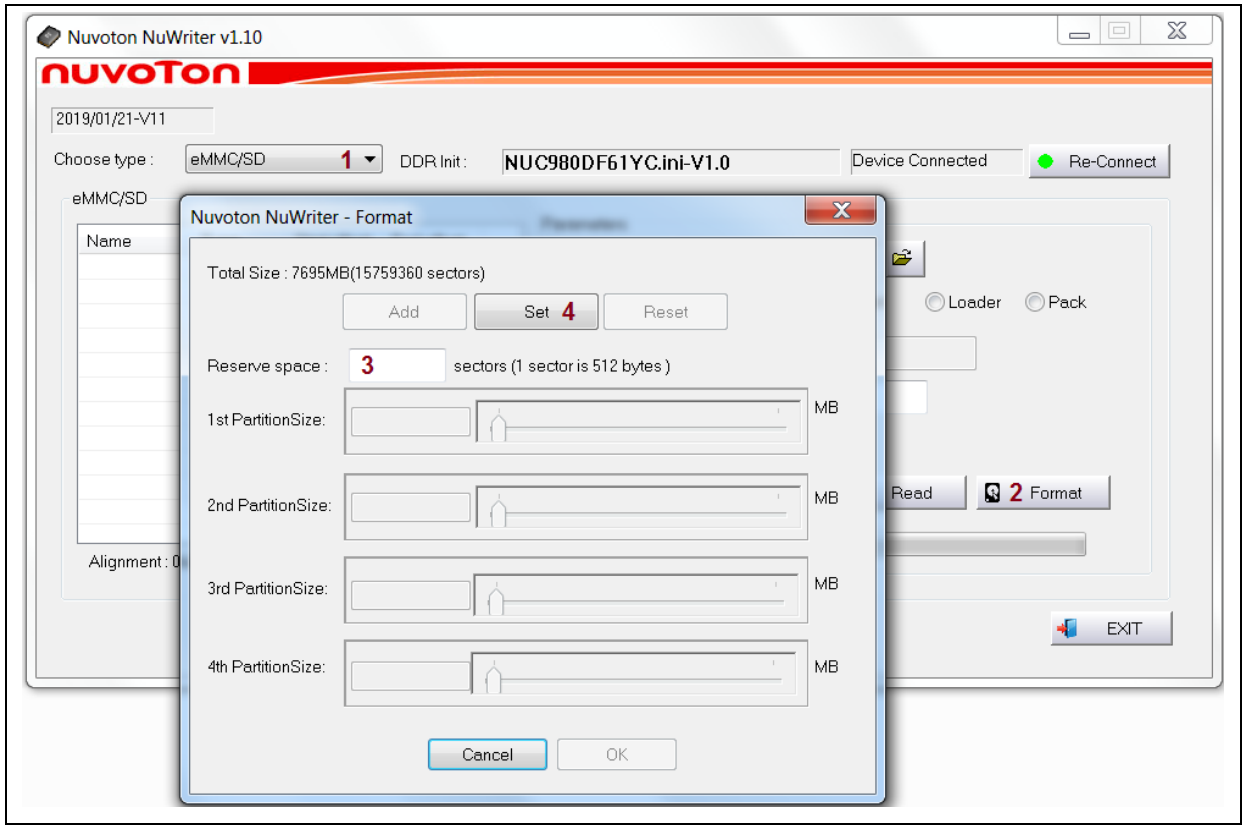


Figure 4-26 eMMC/SD– Format(1)

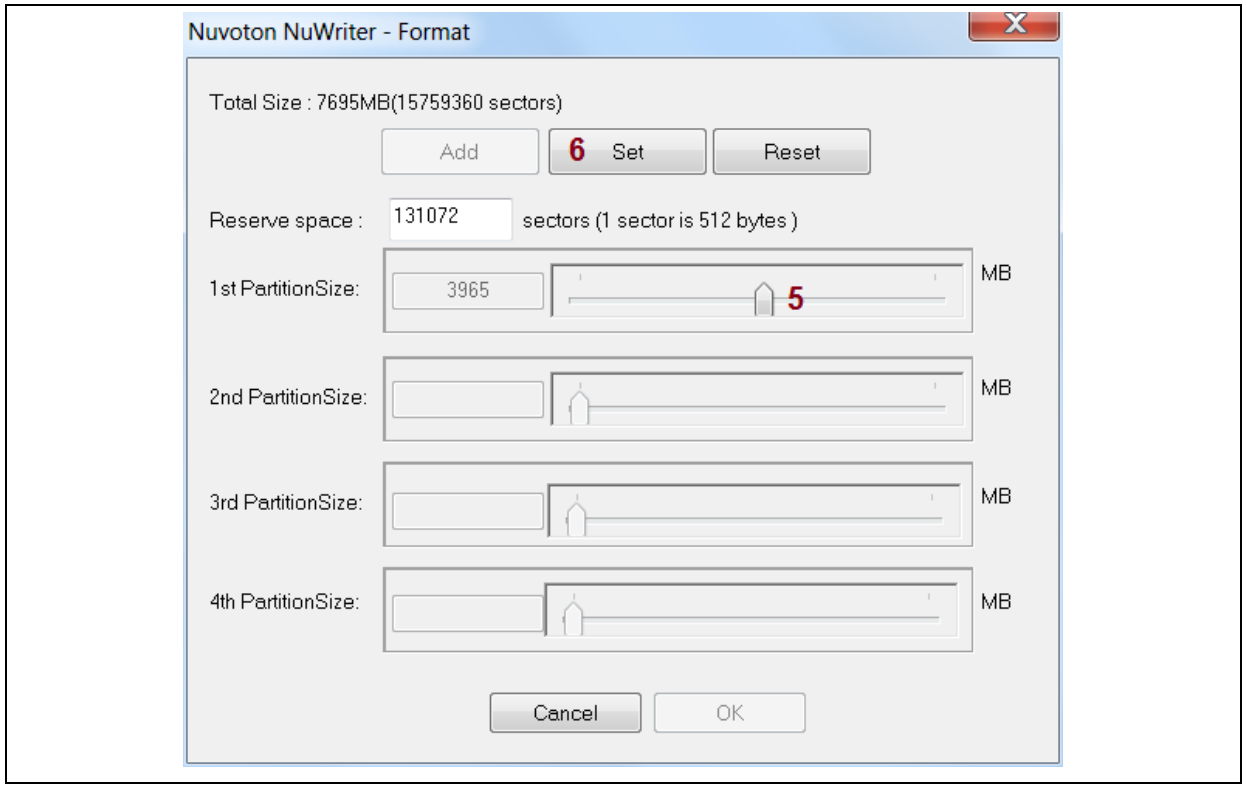


Figure 4-27 eMMC/SD– Format(2)

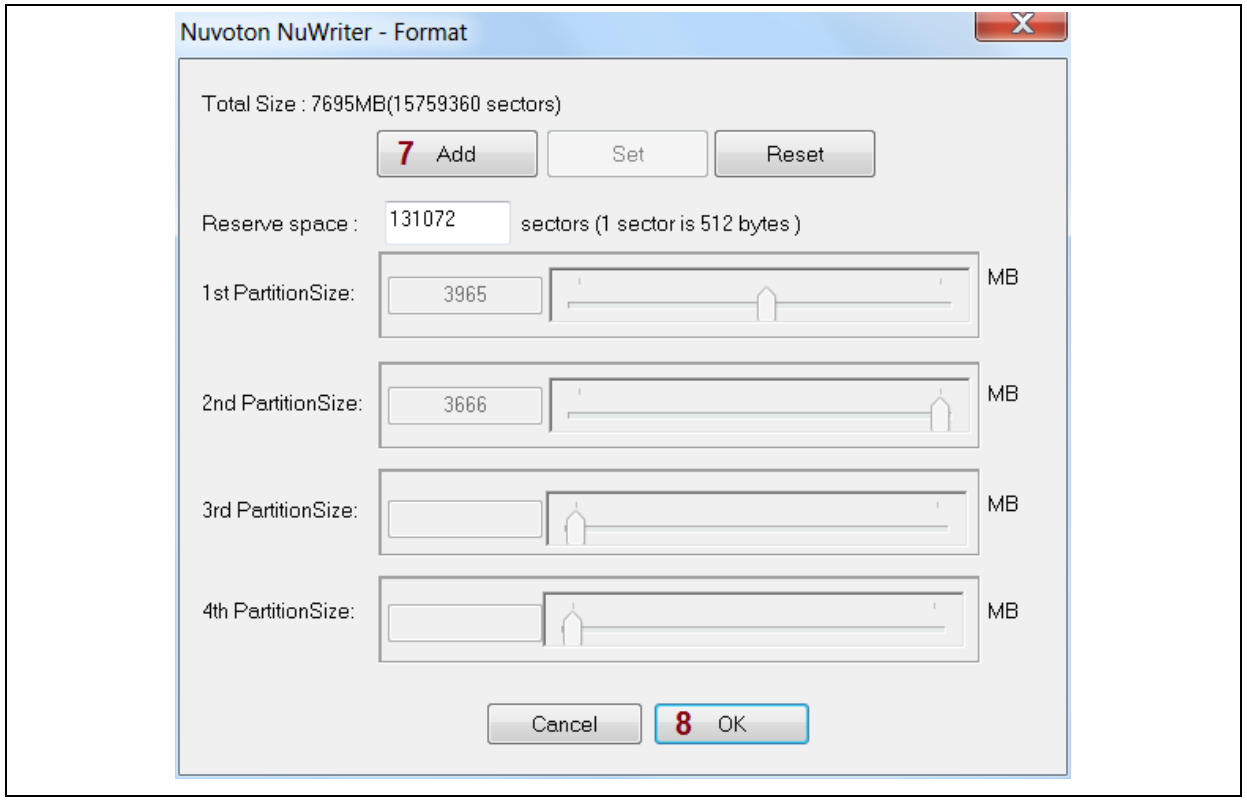


Figure 4-28 eMMC/SD– Format(3)

4.6 SPI NAND Flash模式

SPI NAND模式 可以將Image檔案燒入到SPI NAND Flash中，並且將Image檔案型態設定為Loader 、Data、Environment、Pack，四種型態中的其中一種。

4.6.1 Image檔案型態

4.6.1.1 Loader 型態

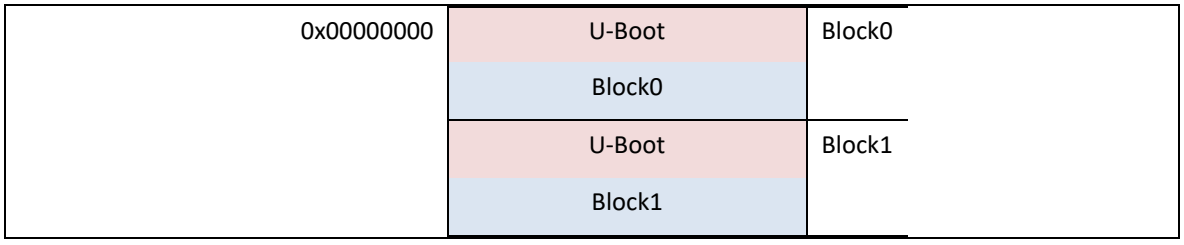
Loader 主要是作為開機的第一支程式，Loader分成Main U-Boot和SPL U-Boot兩個部分，Main U-Boot是完整功能的U-Boot，SPL U-Boot將Main U-Boot從NAND/SPI NAND Flash搬到 DDR 執行。SPL U-Boot只有NAND和SPI NAND boot時才會用到；SPI 開機或eMMC/SD 開機只需要Main U-Boot。SPL U-Boot 的鏈結位址預設為0x200，Main U-Boot的鏈結位址是0xE00000。當NUC980系列晶片上電後會讀取Loader型態的Image並執行，不限定一定要存放Loader，一般程式也是可以使用Loader型態來作為上電後第一支執行程式。Loader Image增加32bytes的檔頭(Header)和DDR參數合併而成，Figure 4-29為Loader型態的Image格式。

	0x0		0x4				0x8				0xC
0x00	Boot Code Marker		Execute Address				Image Size				Reserved
0x10	Page Size	Spare Area	Quad Read cmd	Read Status cmd	Write Status cmd	Status Value	Dummy Byte	Reserved	Reserved	Reserved	Reserved
0x20	DDR - Initial Marker		DDR - Counter				DDR - Address 0				DDR – Value0
0x30	DDR - Address 1		DDR - Value 1				DDR - Address 2				DDR - Value 2
0x40	DDR - Address 3		DDR - Value 3				DDR - Address 4				DDR - Value 5
0x50	DDR - Address 5		DDR - Value 5				DDR - Dummy				DDR - Dummy
0x60	uBoot										

Figure 4-29 Boot Code Header

詳細的內容可以參考NAND Flash模式4.3.1.1章節。

Figure 4-30為Loader image燒入到SPI Nand Flash 中Block0，Block1，Block2，Block3的示意圖：



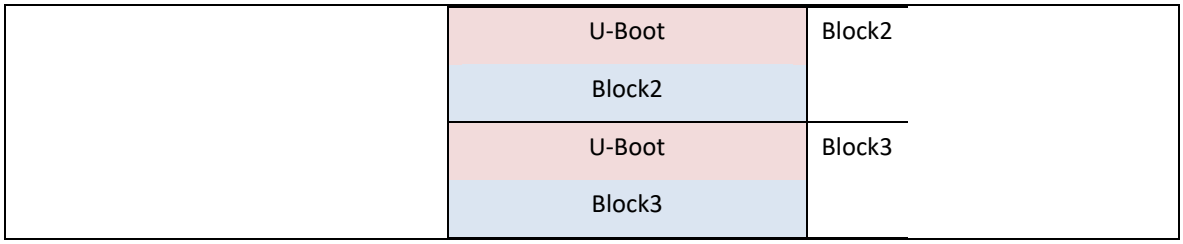


Figure 4-30 Block 0~3 of SPI NAND Flash

4.6.1.2 Data型態

將指定的檔案放入到SPI NAND Flash中所指定的位址，不做任何其他的處理。依據輸入的Image start offset的值(需要對齊Page Size，Page Size是依據SPI NAND Flash規格所決定)，決定將Data存放在SPI NAND Flash的哪個位址，如果Image start offset = 0x10000，則將Data 存放到 SPI NAND Flash 0x10000 的位址，此方式主要可以幫助使用者依據個人需求配置SPI NAND Flash，例如：當Linux Kernel 需要由Loader型態中Image帶起來，Image設定帶起來的位址為0x8000，則需要將Linux Kernel 當作Data型態，配置到0x8000的位址。

4.6.1.3 Environment 型態

主要設定Loader 的環境變數，檔案放入到SPI NAND Flash中所指定的位址，讓Loader 讀取時做相對應的動作。依據輸入的Image start offset的值(需要對齊Block size)，決定將Enviroment存放在SPI NAND Flash的哪個位址，如果Image start offset = 0x10000，則將Enviroment存放到SPI NAND Flash 0x10000 的位址。

4.6.1.4 Pack 型態

依據輸入Pack.bin 的內容，決定將Pack 內容中的Image 放到相對應SPI NAND Flash的位址。Pack.bin 的原理可以參考Pack模式4.7章節。

4.6.1.5 壞塊處理

SPI NAND Flash支援四種型態的方式燒入到SPI NAND Flash中，對於壞塊的寫入處理直接跳過。假設有一資料將寫到NAND Flash的Block 0, Block 1, Block 2, Block 3和Block 4，但是當中的Block 3偵測為壞塊時，會跳過Block 3的寫入處理，往下一個Block繼續寫入。也就是說，寫入的Block變成Block 0, Block 1, Block 2, Block 4和Block 5。

4.6.2 操作方法

4.6.2.1 新增Image

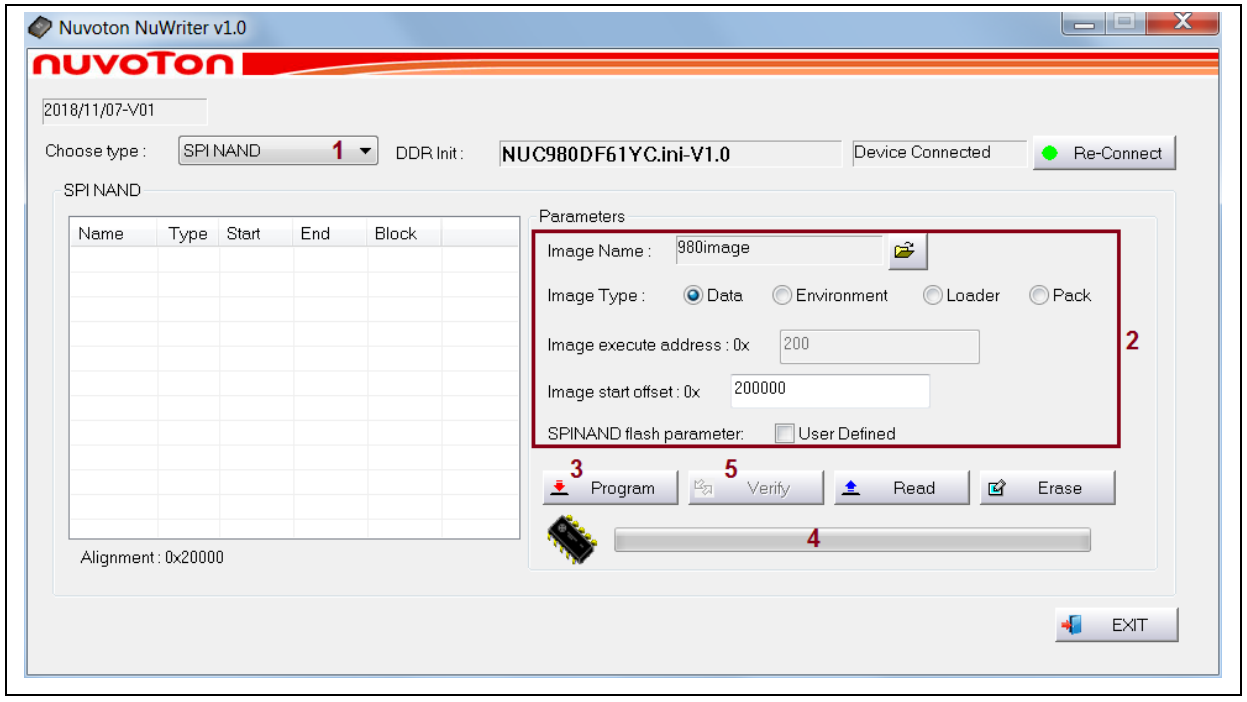


Figure 4-31 SPI NAND – New Image

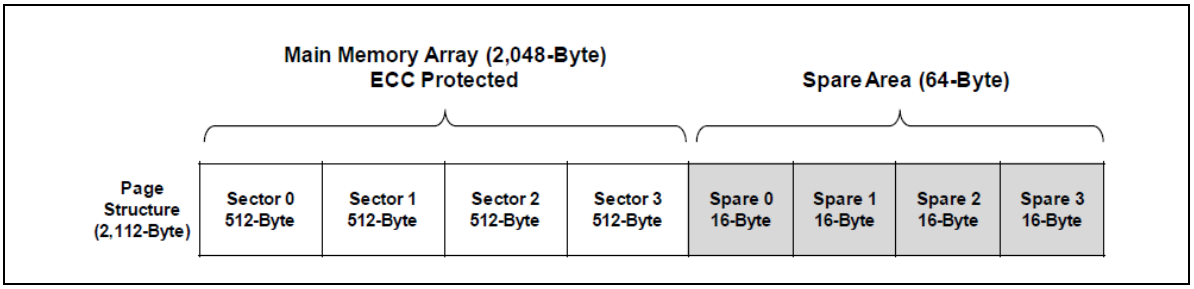
依照Figure 4-31步驟即可以完成新增Image檔案：

1. 選擇“**SPI NAND**”模式， 表格只會紀錄當次燒錄的Image檔案，並不會讀取SPI NAND Flash中Image的資料。
2. 輸入Image檔案資料：
   - Image Name 選擇要燒錄的 Image檔案
   - Image Type 選擇Image型態
   - Image execute address 設置Image執行位置, 依編譯設定而輸入，只有在Loader 型態才有效.
   - Image start offset 設置Image燒錄在SPI NAND Flash的位址。
   - SPI NAND flash parameter 勾選 “**User Defined**”會跳出視窗，提供SPI NAND參數設定。
3. 按下”**Program**”。
4. 等待進度表完成，表格將會顯示這次燒錄完成的Image檔案。
5. 燒錄完成後，可以選擇按下“**Verify**”即可確認燒入資料是否正確。

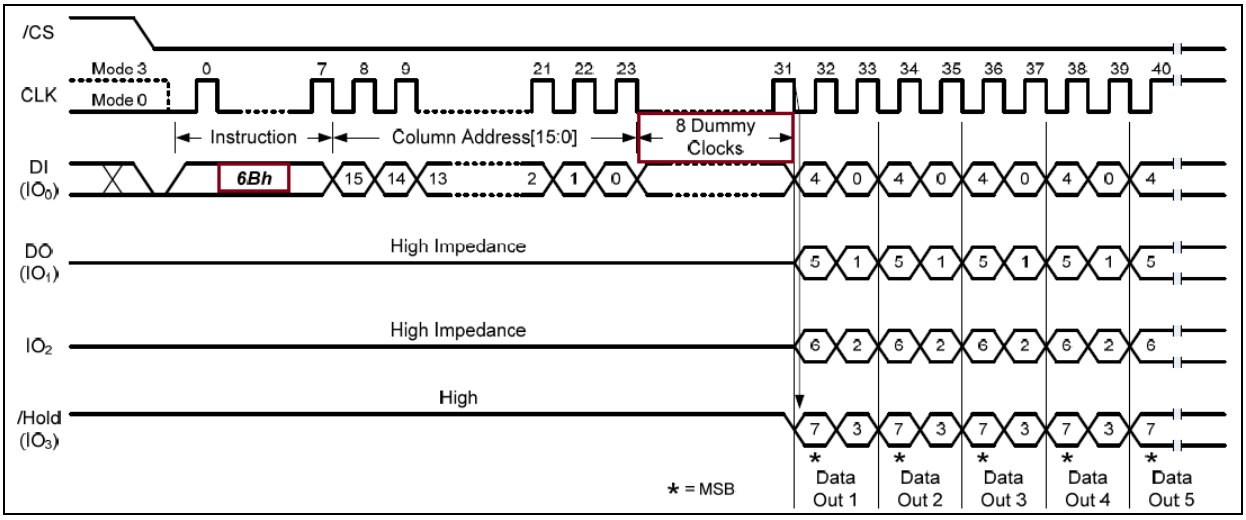
4.6.2.2 自訂SPI NAND參數

NuWriter會自動偵測SPI NAND的ID來決定它的參數，使用者也可以自己依據SPI NAND Flash規格透過NuWriter自行設定參數。使用者可以自訂SPI NAND的Page Size、Spare Area大小、Block Per Flash，以及Page Per Block；此外，也可以設定SPI Quad模式讀取命令、讀取 SPI Flash狀態寄存器命令、寫入 SPI Flash狀態寄存器命令、Quad 模式數值，以及 dummy 位元個數。

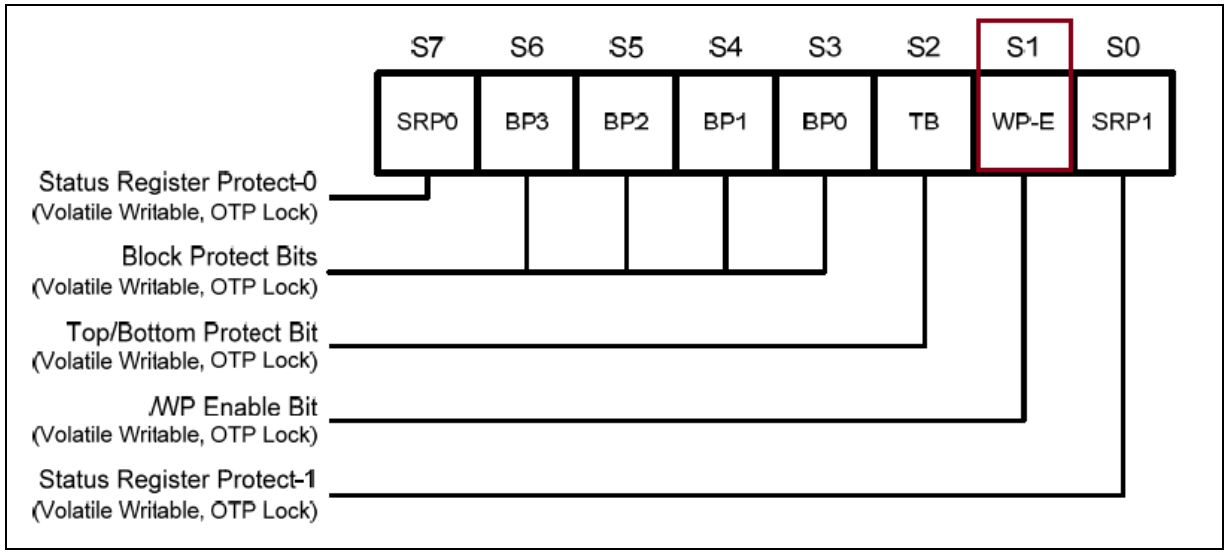
SPI NAND的Page Size、Spare Area大小、Block Per Flash，以及Page Per Block 大小會隨著不同的SPI NAND Flash 型號而不同，相關的資料可以從SPI NAND Flash 規格書中獲得。以下為 SPI NAND Flash 規格書中描述 Page 大小以及 Spare Area 大小的範例。



Quad 模式格式為: QuadReadCmd=Quad模式命令，ReadStatusCmd=讀取狀態寄存器命令，WriteStatusCmd=寫入狀態寄存器命令，StatusValue=狀態寄存器中Quad模式狀態所在的位置，DummyByte= Quad 命令所需的 Dummy Byte個數，命令數值是採用 16 進制。SPI Quad 模式讀取命令同樣也可以從規格書中取得。



SPI Quad模式需要把狀態寄存器中的WP-E位元設置為1，此值對應Nuwriter的StatusValue，狀態寄存器的位元的定義需參考SPI NAND Flash規格書，下圖是從某SPI NAND Flash規格書裡所擷取。



依照Figure 4-32步驟即可以完成設定SPI NAND參數：

1. 選擇 “**SPI NAND**” 模式。
2. 勾選 “**User Defined**”會跳出視窗，提供SPI NAND參數設定。

3. 根據SPI NAND Flash規格輸入Page Size、Spare Area、Quad Read Command、Read Status Command、Write Status Command、Status Value、Dummy Byte、Block Per Flash及Page Per Block。
4. 按下 “**OK**”， 即可完成參數設定。

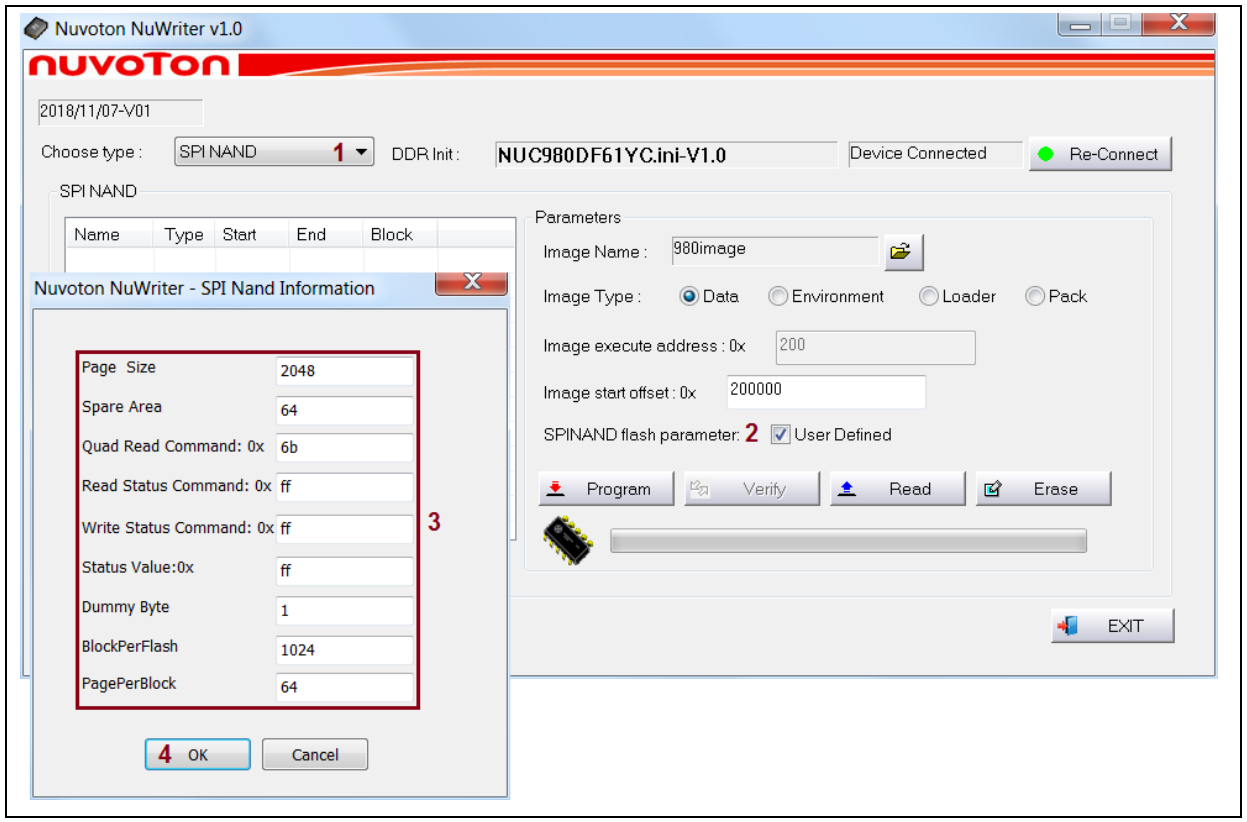


Figure 4-32 SPI NAND – Configure Parameters

4.6.2.3 讀取Image

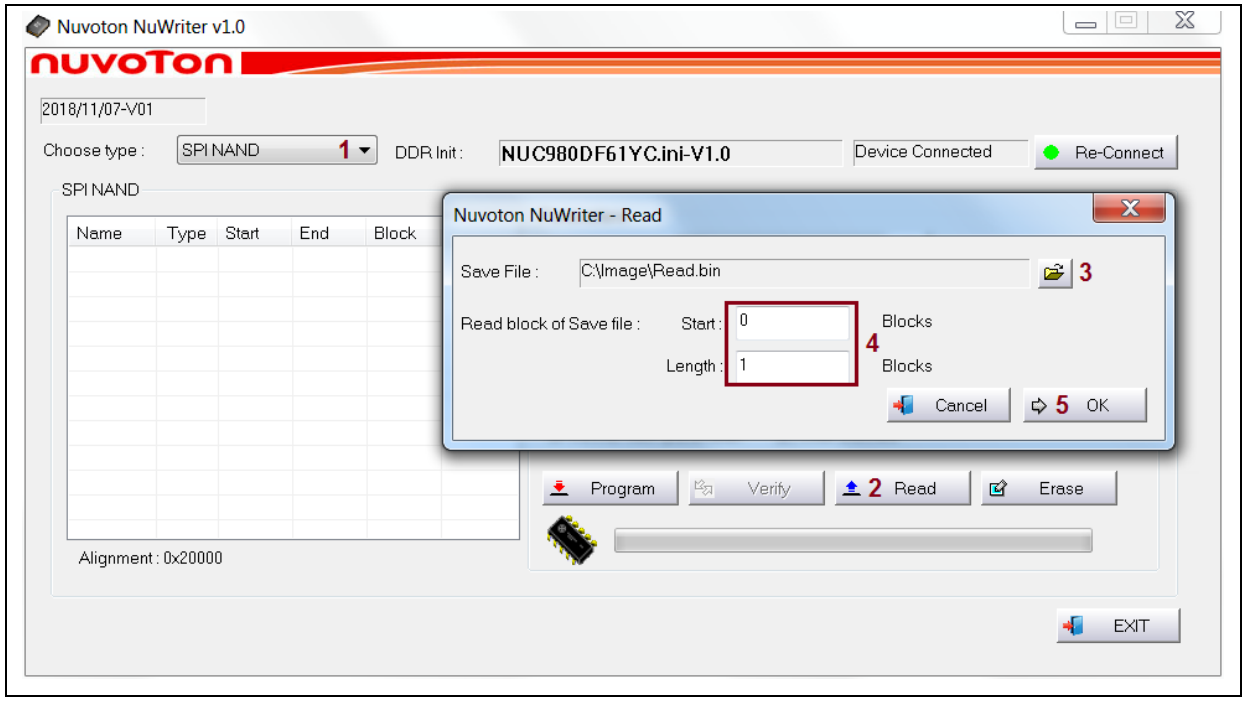


Figure 4-33 SPI NAND – Read

依照Figure 4-33步驟即可以完成讀取Image：

1. 選擇 “**SPI NAND**” 模式。
2. 按下 “**Read**”。
3. 選擇要儲存檔案的位置。
4. 輸入欲讀取的Block起始位置與Block長度(每一Block大小依據SPI NAND Flash規格來決定) 。
5. 按下 “**OK**”， 即可完成Image讀取。

4.6.2.4 移除 Image

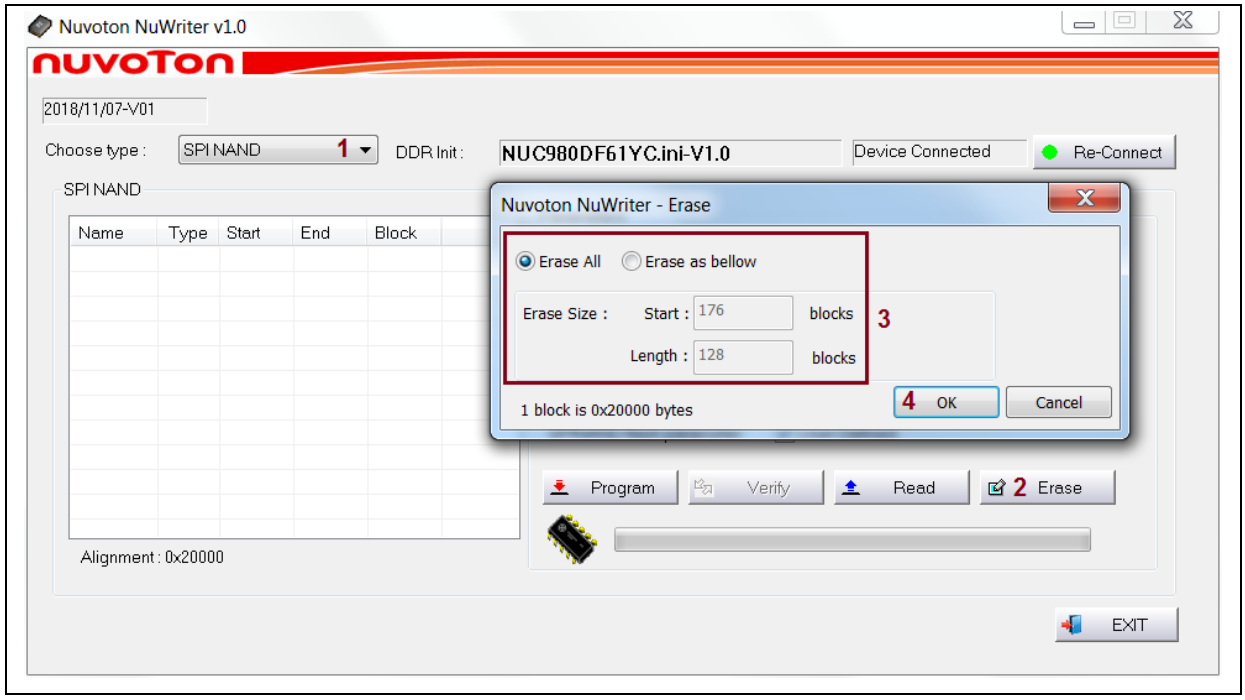


Figure 4-34 SPI NAND – Erase

依照Figure 4-34步驟即可以完成移除Image檔案：

1. 選擇 “**SPI NAND**” 模式 。
2. 按下 “**Erase**”
3. 選擇 “**Erase All**” 或選擇 “**Erase as below**”
   - 輸入擦除的Blocks(每一Block大小依據SPI NAND Flash規格來決定)
4. 按下 “**OK**”，即可完成擦除。

4.7 Pack 模式

將每個Image檔案合併成一個Pack.bin檔案，若是Image檔案類別為Loader，則需要在前面加32bytes的檔頭及DDR參數再與Loader檔案合併，Pack.bin檔案格式如下：

	0x0	**0x0**	**0x0**	**0x0**
0x00	Initial Marker	File Length	File Number	Reserve
0x10	Child0-File Length	Child0-File Address	Child0-Image Type	Child0-Reserve
	Child0-data			
	Child1-File Length	Child1-File Address	Child1-Image Type	Child1-Reserve
	Child1-data			

	Child2-File Length	Child2-File Address	Child2-Image Type	Child2-Reserve
	Child2-data			

Figure 4-35 Pack Header/Pack Child Header

Initial Marker = {0x00000005}。

File Length = {所有Image的長度}，對齊64Kbytes。

File Number = {有多少個Image檔}。

Child0-File Length = {第一個Image 資料的長度}。

Child0-File Address = {第一個Image燒入的位址}。

Child0-Image Type = {第一個Image的類別}。

Child0-data = = {第一個Image 資料}。

Child1-File Length = {第二個Image 資料的長度}。

Child1-File Address = {第二個Image燒入的位址}。

Child1-Image Type = {第二個Image的類別}。

Child1-data = = {第二個Image 資料}。

Figure 4-36假設在NuWriter中 Pack 模式輸入兩個Image檔案分別為u-boot-spl.bin 和 u-boot.bin。

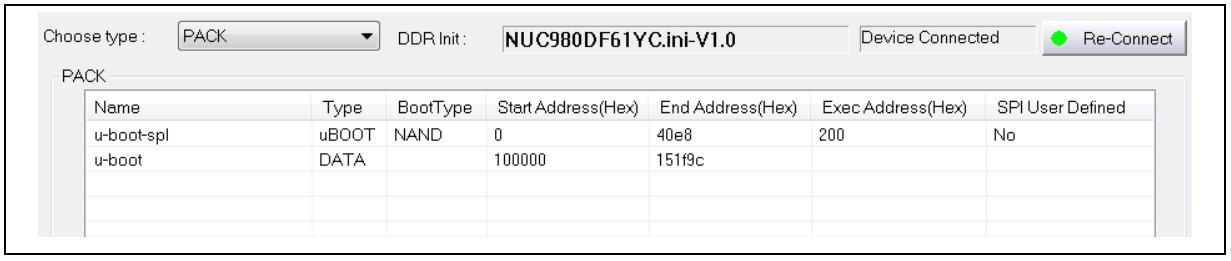


Figure 4-36 Pack Image

可以得知完成後的Pack格式如下：

Initial Marker = 0x00000005。

File Length = (0x000040e8+0x00051f9c) ≅ 0x00070000。

File Number = 2。

Child0-File Length = 0x000040e8。

Child0-File Address = 0。

Child0-Image Type = 0x2。

Child0-data = { address 0x00000000 + 0x000001c0 ~ 0x000001c0 + 0x000040e8}。

Child1-File Length =0x51f9c。

Child1-File Address =0x100000。

Child1-data = { address 0x00004108 + 0x0000010 ~ 0x0004118 + 0x00051f9c }。

```
Address   0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
                Initial Marker    File Length       File Number       Reserved
00000000  05 00 00 00 00 00 07 00 02 00 00 00 ff ff ff ff
                Child0-File Length  Child0-File Address  Child0-Image Type  Child0-Reserved
00000010  e8 40 00 00 00 00 00 00 02 00 00 00 ff ff ff ff
                32Byte Header and DDR Parameter for uBoot Image
00000020  20 54 56 4e 00 02 00 00 48 3f 00 00 ff ff ff ff
00000030  00 08 40 00 6b 35 31 02 01 ff ff ff ff ff ff ff
00000040  55 aa 55 aa 2e 00 00 00 64 02 00 b0 18 00 00 c0
00000050  20 02 00 b0 18 00 00 01 aa 55 aa 55 01 00 00 00
00000060  aa 55 aa 55 01 00 00 00 28 20 00 b0 4a d8 dc 53
00000070  08 20 00 b0 14 80 00 00 aa 55 aa 55 01 00 00 00
00000080  aa 55 aa 55 01 00 00 00 aa 55 aa 55 01 00 00 00
00000090  00 20 00 b0 7e 04 03 00 aa 55 aa 55 01 00 00 00
000000a0  04 20 00 b0 21 00 00 00 aa 55 aa 55 01 00 00 00
                ...
000001a0  aa 55 aa 55 01 00 00 00 aa 55 aa 55 01 00 00 00
000001b0  aa 55 aa 55 01 00 00 00 00 00 00 00 00 00 00 00
                ...
000040f0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
                                  Child1-File Length  Child1-File Address
00004100  00 00 00 00 00 00 00 00 9c 1f 05 00 00 00 10 00
                Child1-Image Type Child1-Reserved
00004110  00 00 00 00 ff ff ff ff be 00 00 ea 14 f0 9f e5
00004120  14 f0 9f e5 14 f0 9f e5 14 f0 9f e5 14 f0 9f e5
00004130  14 f0 9f e5 14 f0 9f e5 60 00 e0 00 c0 00 e0 00
```

Figure 4-37 Output Pack.bin

4.7.1 Image檔案型態

Pack模式提供Loader、Data、Environment及eMMC/SD Partition四種Image檔案型態，其中eMMC/SD Partition檔案型態用來記錄使用者自行配置eMMC/SD的分割區大小資訊，此種檔案型態只有eMMC/SD格式化才會使用。eMMC/SD Partition檔案型態由48 bytes的檔頭(Header)組合而成，Figure 4-38為eMMC/SD Partition型態的Image格式。

	0x0	0x4	0x8	0xC
0x00	Reserved	Partition Number	Reserved Space Size	Reserved
0x10	Partition1 Size	Partition1 Sector Size	Partition2 Size	Partition2 Sector Size
0x20	Partition3 Size	Partition 3 Sector Size	Partition4 Size	Partition4 Sector Size

Figure 4-38 Pack Mode eMMC/SD Partition Header

4.7.2 操作方法

Pack模式可以將多個Image檔案合併成一個Pack Image檔案，往後就可利用NuWriter將這個Pack Image直接快速的燒入到對應的儲存體(eMMC/SD、NAND Flash、SPI NOR/NAND Flash)。

4.7.2.1 新增Image

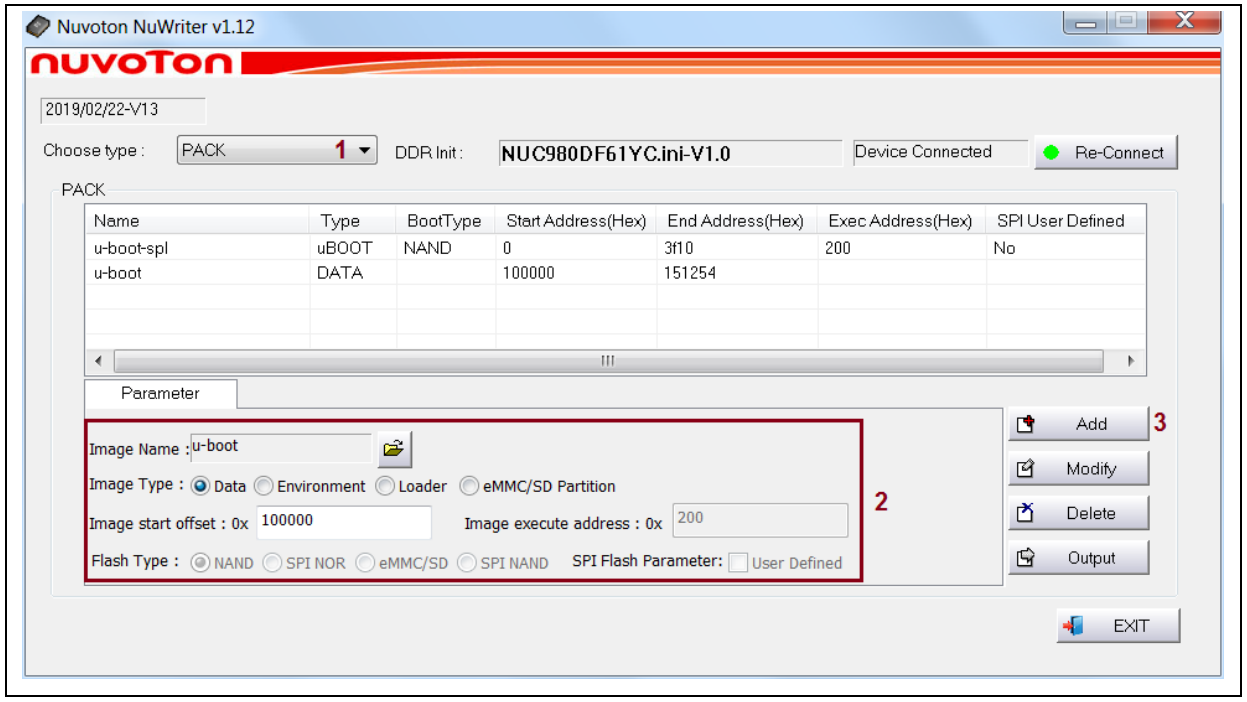


Figure 4-39 PACK – New Image

依照Figure 4-39步驟即可以完成新增image檔案：

1. 選擇“**PACK**” 模式。
2. 輸入Image檔案資料：
   (1) Image Type: Data、Environment或Loader
   - Image Name 選擇要燒錄的檔案
   - Image Type 選擇Image型態。
   - Image execute address設置Image執行位置, 依編譯設定而輸入，只有在Loader型態才有效
   - Image start offset 燒錄起始位置
   - Flash Type選擇Flash類型，只有在Loader與Environment型態才有效
   - SPI Flash Parameter提供給Flash Type為SPI NOR或是SPI NAND類型且Image Type為Loader 型態才有效。

   (2) Image Type: eMMC/SD Partition
   - Image Type選擇eMMC/SD Partition型態，會跳出eMMC/SD格式化視窗提供使用者自行配置eMMC/SD的空間，設定方式可以參考**Error! Reference source not found.**。
   - Image Name、Image execute address、Image start offset、Flash Type、 SPI Flash Parameter不需要輸入資料。
3. 按下“**Add**”按鈕後，上方表格將顯示新增檔案的資訊。Name欄位為Image檔案名稱，若是Image型態為eMMC/SD Partition，此欄位會顯示”Partition_INFO”。Type欄位為Image型態，若是Image型態為eMMC/SD Partition，此欄位會顯示”FORMAT”。BootType欄位為Flash類

型。Start Address(Hex)欄位為燒錄起始位置。Exec Address(Hex)欄位為執行位置，以16進制顯示，SPI User Defined欄位以Yes或No代表是否自訂SPI參數。

4.7.2.2 修改Image

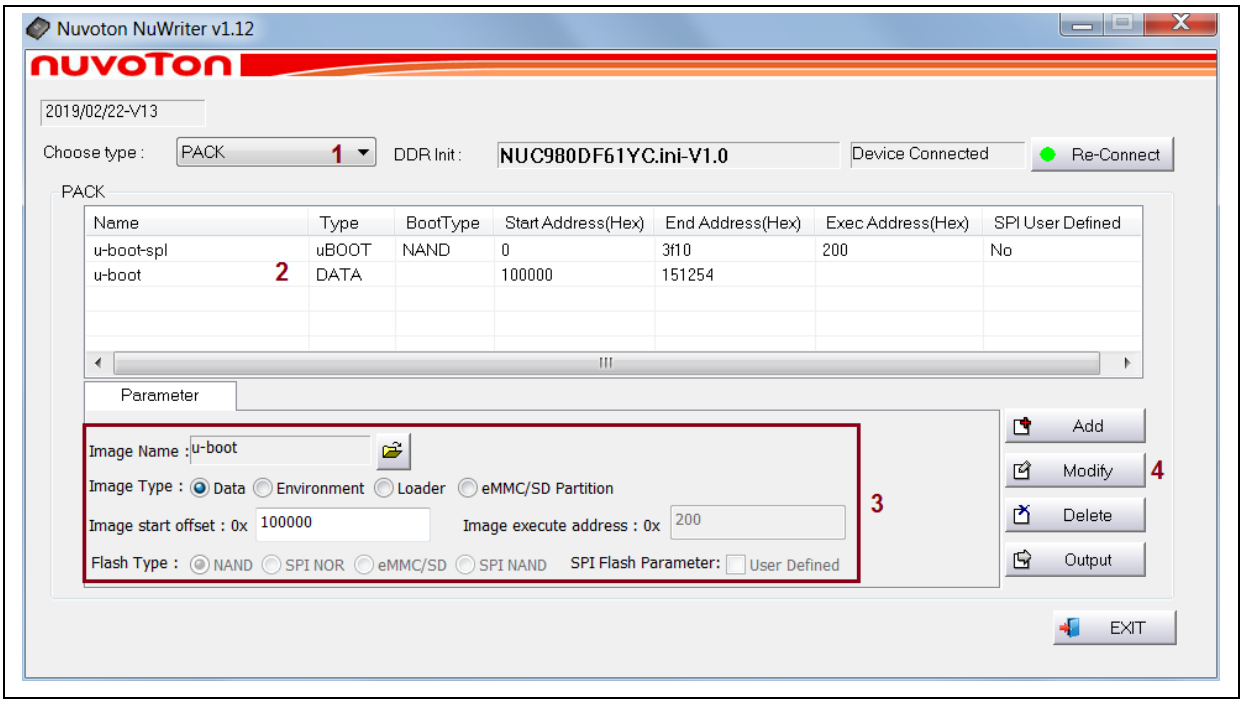


Figure 4-40 PACK – Modify Image

Figure 4-40步驟即可以完成修改image：

1. 選擇“**PACK**”模式。
2. 在表格上面按一下需要修改的Image。
3. 修改Image檔案資料：
   - Image Name 選擇要燒錄的檔案
   - Image Type 選擇Image型態
   - Image execute address 設置Image執行位置, 依編譯設定而輸入，只有在Loader 型態才有效
   - Image start offset 燒錄起始位置
   - Flash Type選擇flash類型，只有在Loader與Environment型態才有效
   - SPI Flash Parameter提供給Flash Type為SPI NOR或是SPI NAND類型且Image Type為Loader 型態才有效。
4. 按下“**Modify**”按鈕。

4.7.2.3 移除Image

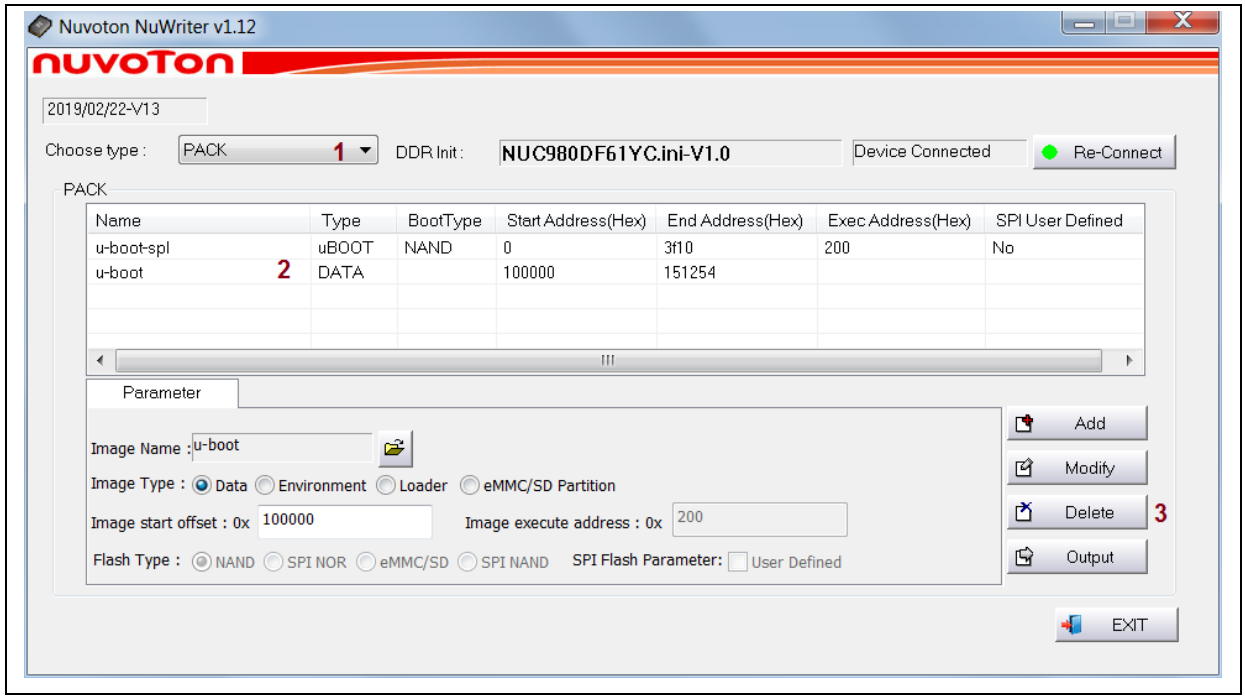


Figure 4-41 PACK – Delete Image

依照Figure 4-41步驟即可以完成移除Image：

1. 選擇 “**PACK**” 模式。
2. 在表格上點一下要刪除的Image。
3. 按下“**Delete**”按鈕，即可移除Image。

4.7.2.4 輸出Pack檔案

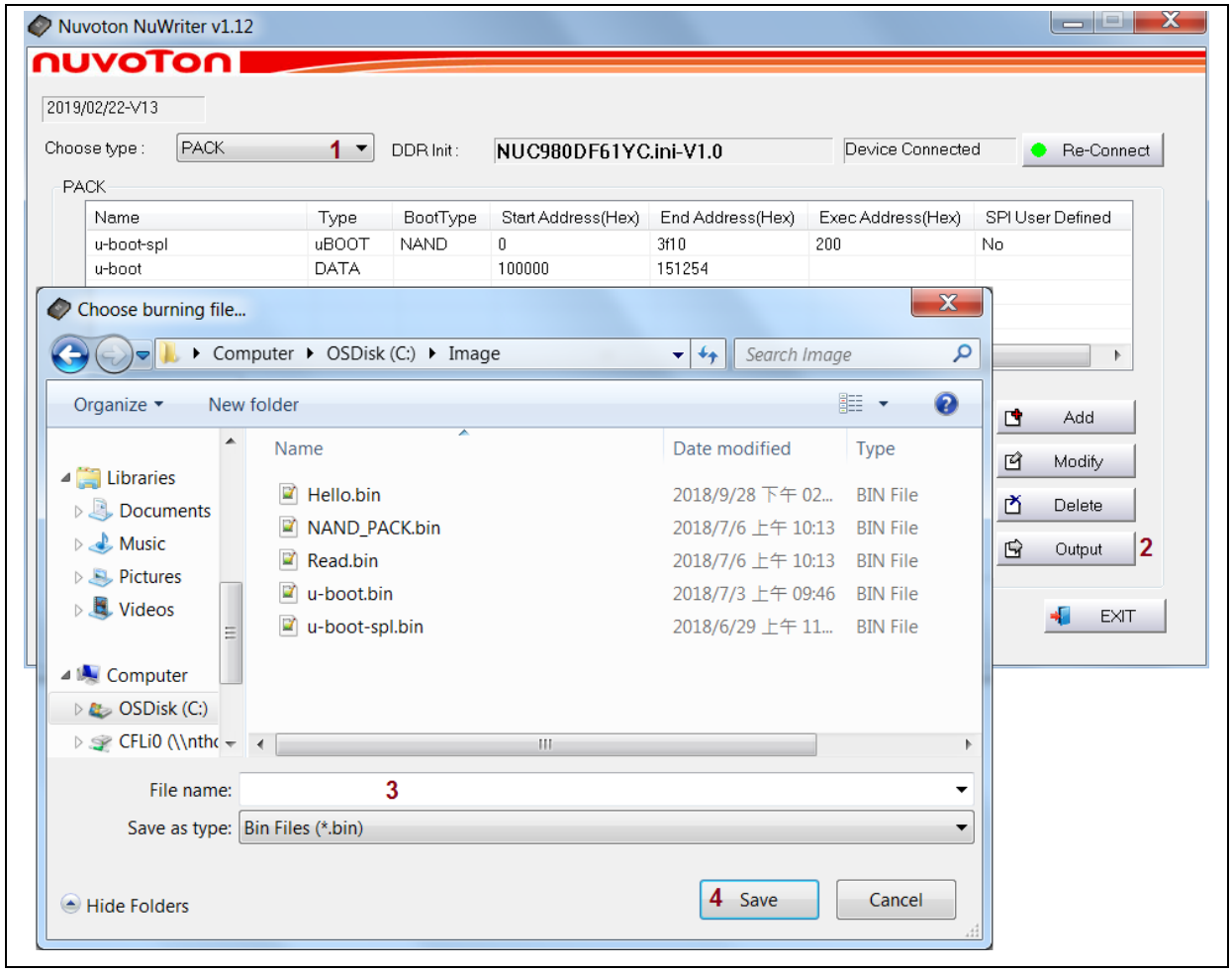


Figure 4-42 PACK – Output Pack Image

依照Figure 4-42步驟即可以完成輸出Pack檔案：

1. 選擇 “**PACK**” 模式。
2. 按下 “**Output**”。
3. 選擇欲儲存的檔案。
4. 按下”**Save**”按鈕即可產生Pack Image。

4.7.2.5 燒錄Pack檔案

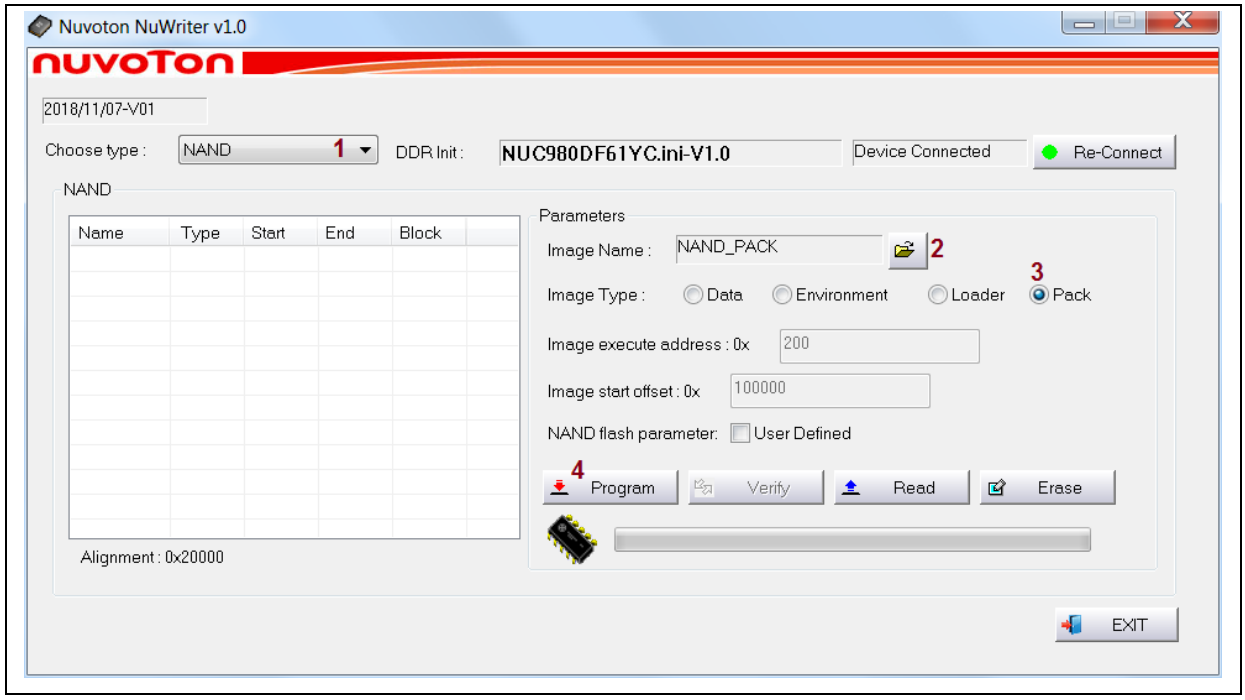


Figure 4-43 PACK – Burn Pack Image

依照Figure 4-43步驟以NAND Flash完成燒錄Pack檔案：

1. 選擇需要燒錄的儲存體(eMMC/NAND Flash/ SPI NOR Flash// SPI NAND Flash)。
2. 在Image Name 選擇要之前所產生的Pack檔案。
3. Image Type選擇”**Pack**” 型態。
4. 按下 “**Program**” 按鈕，即可完成燒錄。另外，Image Type為Pack型態燒錄完成後無法使用 “**Verify**” 按鈕確認燒入資料是否正確。

4.7.3 製作和燒錄pack範例

本章節將示範如何將*nand_env.txt*、*u-boot.bin*和*u-boot-spl.bin*製作成一個Pack Image並燒錄到NAND Flash。Pack Image將由下列三個檔案合併而成：

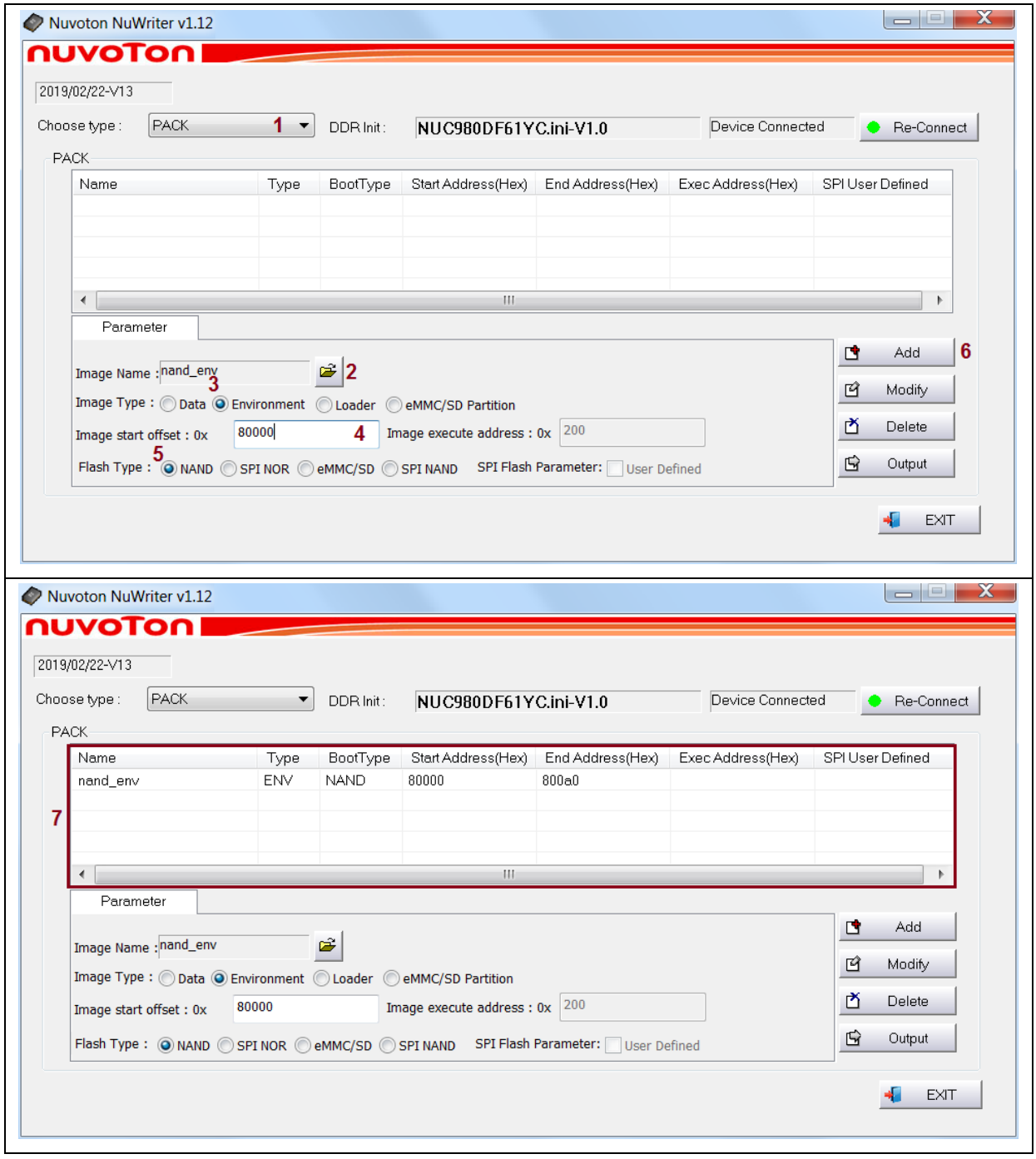
1. *nand_env.txt*: U-Boot 環境變數(預設 offset為 0x80000) 。
2. *u-boot.bin*: 主要的 U-Boot (執行位置為 0x100000) 。
3. *u-boot-spl.bin*: SPL U-Boot (預設DDR執行位置為 0x200) 。

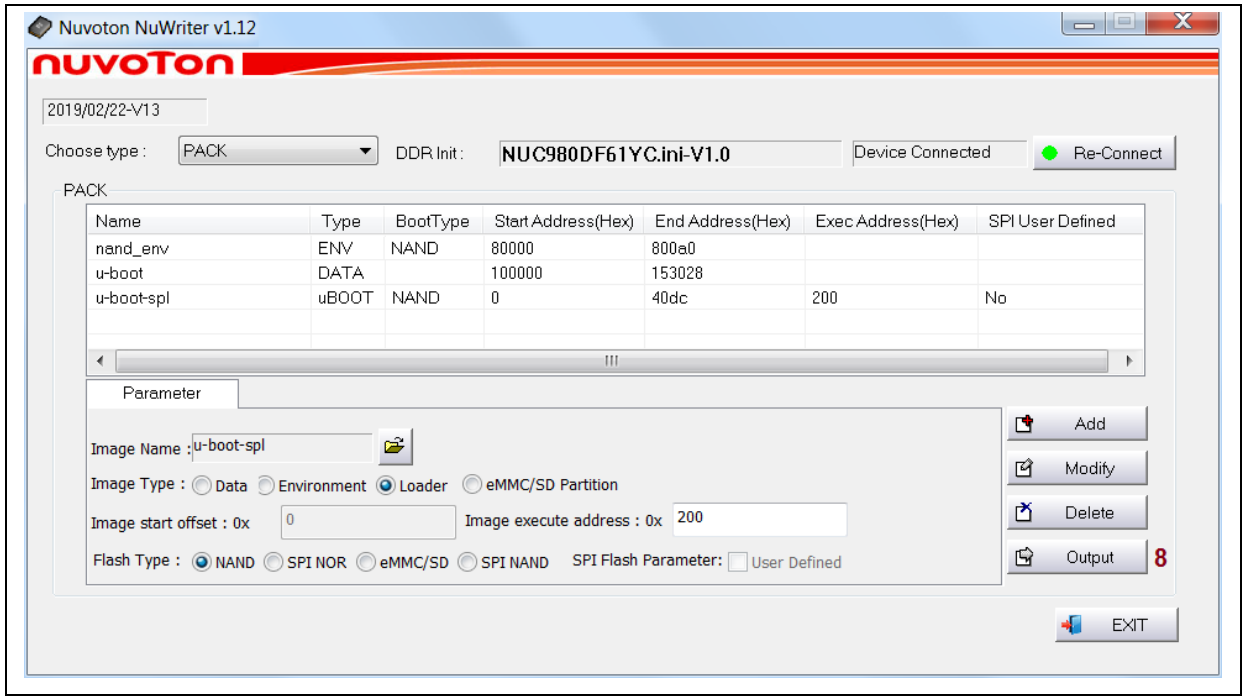
依照下列步驟可將多個Image合併製作為一個Pack Image。

1. 選擇”**PACK”**模式。
2. 選取檔案*nand_env.txt*。
3. Image Type設定為Environment。
4. Image start offset設定為0x80000。
5. 按下”**Add**”按鈕。
6. 成功加入Image後，可以在上方的表格中看到剛才加入的Image相關資訊。

重複上述六個步驟，將其他Image依序加入。

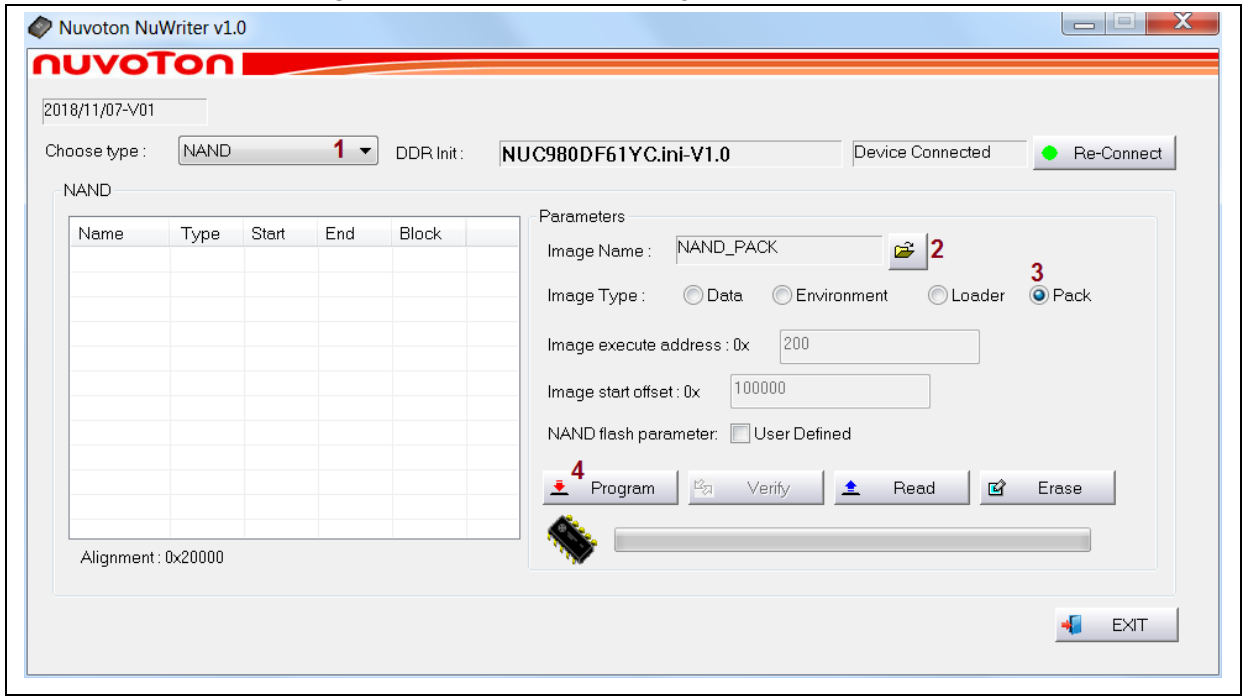
7. 最後，按下”**Output**”按鈕即可產生一個Pack檔案。


Nuvoton NuWriter v1.12
2019/02/22-V13
Choose type :
PACK
1
DDR Init :
NUC980DF61YC.ini-V1.0
Device Connected
Re-Connect
PACK
Name
Type
BootType
Start Address(Hex)
End Address(Hex)
Exec Address(Hex)
SPI User Defined
Parameter
Image Name :
nand_env
3
2
Image Type :
Data
Environment
Loader
eMMC/SD Partition
Image start offset : 0x
80000
4
Image execute address : 0x
200
5
Flash Type :
NAND
SPI NOR
eMMC/SD
SPI NAND
SPI Flash Parameter:
User Defined
Add
6
Modify
Delete
Output
EXIT
Nuvoton NuWriter v1.12
2019/02/22-V13
Choose type :
PACK
DDR Init :
NUC980DF61YC.ini-V1.0
Device Connected
Re-Connect
PACK
Name
Type
BootType
Start Address(Hex)
End Address(Hex)
Exec Address(Hex)
SPI User Defined
nand_env
ENV
NAND
80000
800a0
7
Parameter
Image Name :
nand_env
Image Type :
Data
Environment
Loader
eMMC/SD Partition
Image start offset : 0x
80000
Image execute address : 0x
200
Flash Type :
NAND
SPI NOR
eMMC/SD
SPI NAND
SPI Flash Parameter:
User Defined
Add
Modify
Delete
Output
EXIT



依照下列步驟將Pack Image燒錄至NAND Flash：

1. 選擇”**NAND**”模式。
2. Image Name 選擇剛剛產生出來的Pack Image。
3. Image Type 選擇”**Pack**”。
4. 最後，按下”**Program**”按鈕即可燒錄Pack Image。



4.8 Mass Production 模式

Mass Production模式提供一對多燒錄功能，此功能會自動偵測有多少設備連接到PC，最多可以同時支援八台設備對相同的儲存裝置做燒錄動作。此模式使用的檔案必須是透過Pack模式製造產生出來的檔案，執行時會先把該儲存裝置Erase，接下來才進行Program，最後會再進行資料比對。

4.8.1 操作方法

因為Pack模式已經將所有的Image合併為一個檔案，所以使用者只需要選擇欲燒錄的檔案、儲存到哪種儲存裝置，最後再按下"**Start**"按鍵，就可以同時多台設備燒寫。若是欲燒錄的儲存裝置為NAND，可以選擇使用或不使用自訂參數。

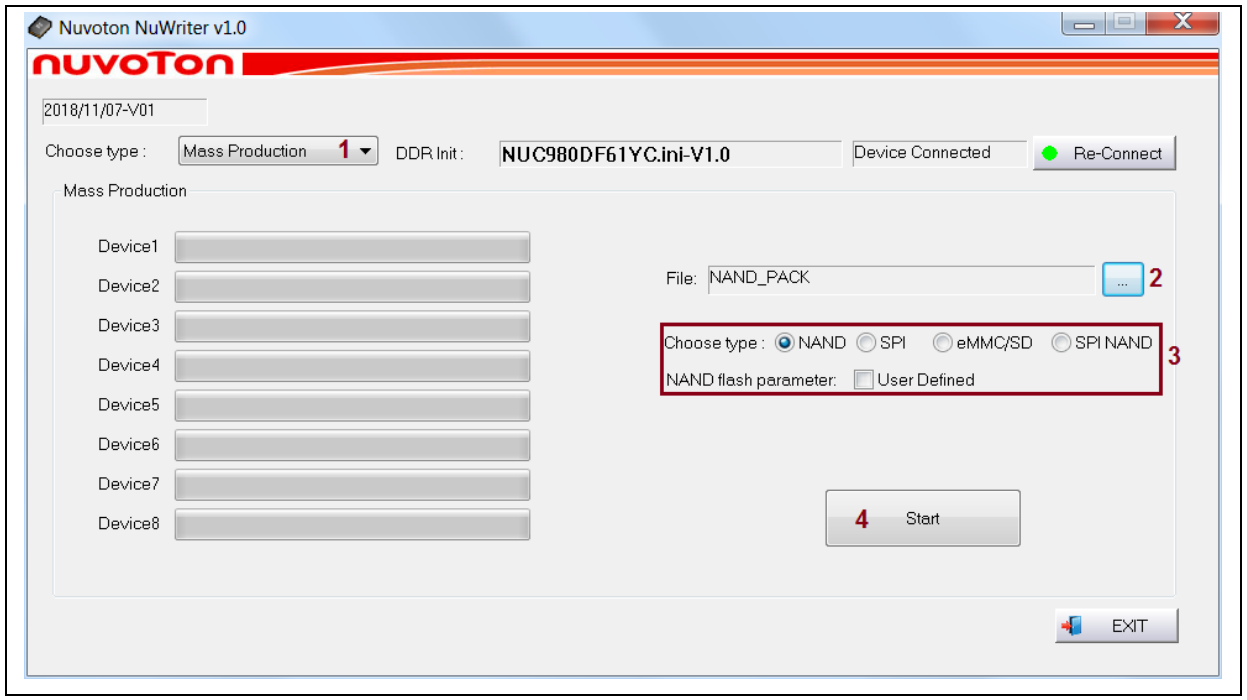


Figure 4-44 Mass Production

5 NUWRITER 原始碼

5.1 Software Code

Nuwriter PC端Tool是使用VS2010為開發環境，所以使用VS2010可以開啟NuWriter專案，開啟後如下圖所示：

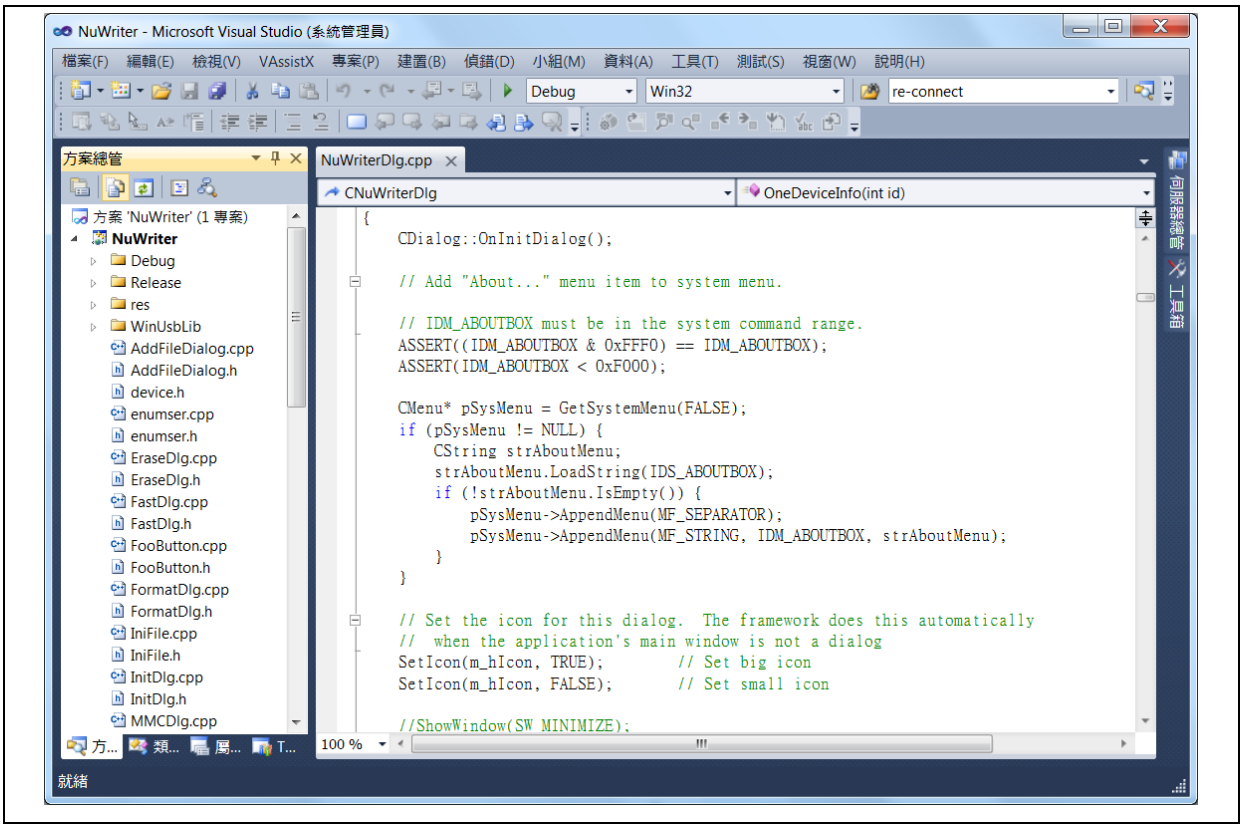


Figure 5-1 NuWriter – Project

可以透過Resource view裡面的Dialog資料夾中看見所有NuWriter的畫面，可以依據使用者需求而進行修改。

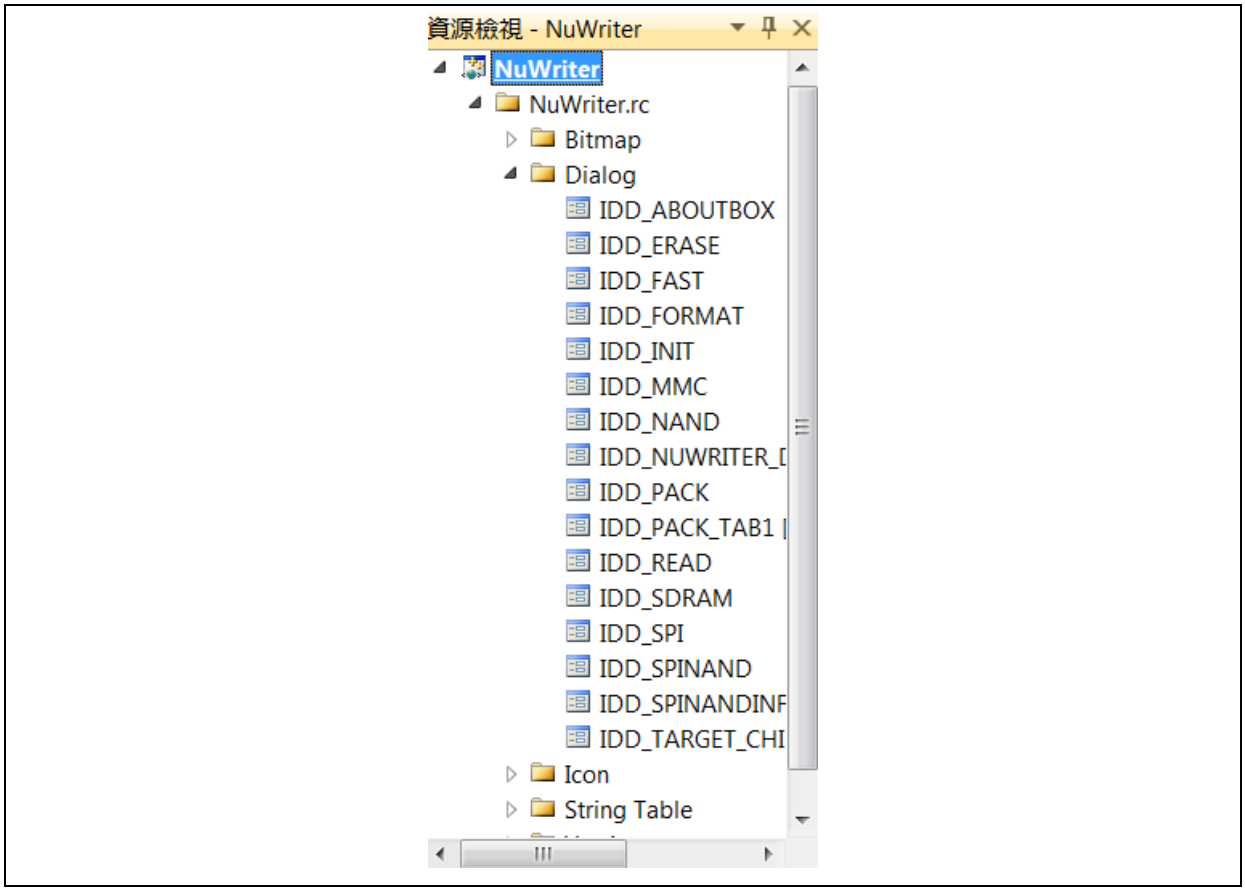


Figure 5-2 NuWriter – Resource View

假設在IDD_SRAM上面點擊兩下，即可看見下面的畫面：

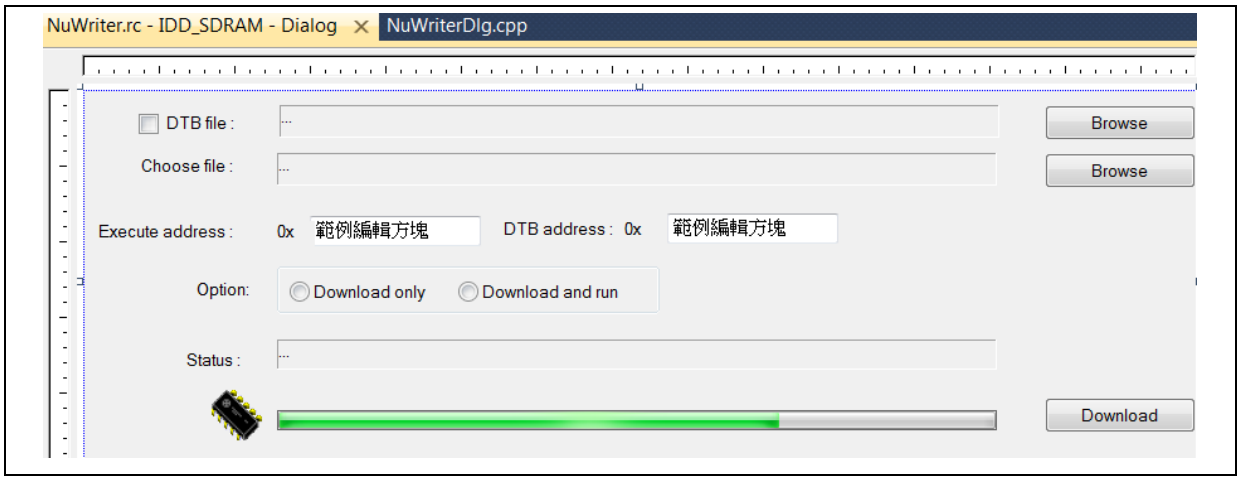


Figure 5-3 IDD_SDRAM Dialog

依序再點擊Download按鈕兩下即可看見Downalod按鈕的程式，當NuWriter運作的時候，按下Download時所運作的程式內容如Figure 5-4：

RAMDlg.cpp | NuWriter.rc - IDD_SDRAM - Dialog | NuWriterDlg.cpp

CRAMDlg | OnBnClickedSdramDownload()

```
void CRAMDlg::OnBnClickedSdramDownload()
{

    CString dlgText;

    UpdateData(TRUE);
    InitFile(1);
    if(m_address.IsEmpty())
    {
        AfxMessageBox(_T("Please input execute address"));
        return;
    }

    GetDlgItem(IDC_SDRAM_DOWNLOAD)->GetWindowText(dlgText);

    if(dlgText.CompareNoCase(_T("Download"))==0)
    {

        UpdateData(TRUE);

        unsigned Thread1;
        HANDLE hThread;
```

Figure 5-4 OnBnClockedSdramDownload Function

在NuWriter 中是由許多的按鈕來完成所有的功能，使用者可以針對對應的按鈕來看source code，進而了解NuWriter的運作方式。

NuWriter 與NuWriterFW之間溝通主要透過NucWinUsb.h和NucWinUsb.cpp來做為之間的橋樑。

NucWinUsb.h ：

```
static const GUID OSR_DEVICE_INTERFACE[] = { 0xD696BFEB, 0x1734, 0x417d,
{ 0x8A, 0x04, 0x86,0xD0,0x10,0x71,0xC5,0x12 } };
typedef struct PIPE_ID
{
      UCHAR  PipeInId;
      UCHAR  PipeOutId;
}Pipe_Id,*PPipe_Id;

typedef struct WINUSBHANDLE
{
      HANDLE hDeviceHandle;
      WINUSB_INTERFACE_HANDLE hUSBHandle;
      Pipe_Id pipeid;
      CString DevicePath;
}SWinUsbHandle,*PSWinUsbHandle;

#define MAX_DEVICE_NUM 32
```

```
class CNucWinUsb
{
      public:
            CNucWinUsb();
            SWinUsbHandle WinUsbHandle[MAX_DEVICE_NUM];
            int WinUsbNumber;
            UCHAR DeviceSpeed;
            /* Enable WinUSB driver */
            BOOL EnableWinUsbDevice(void);
            /* Set mode */
            BOOL NUC_SetType(int id,USHORT type,UCHAR * ack,ULONG len);
            /* Write data to deivce */
            BOOL NUC_WritePipe(int id,UCHAR *buf,ULONG len);
            /* Read data from deivce */
            BOOL NUC_ReadPipe(int id,UCHAR *buf,ULONG len);
            /* Disable WinUSB driver */
            void NUC_CloseHandle(void);
            /* Check whether ID is connected or not */
            BOOL NUC_CheckFw(int id);
      protected:
            BOOL GetDeviceHandle(void);
            BOOL GetWinUSBHandle(void);
            BOOL GetUSBDeviceSpeed(void);
            BOOL QueryDeviceEndpoints(void);
};
static CNucWinUsb NucUsb;
```

假設如果有設備插入到電腦中，可以依序透過下列順序與設備溝通：

```
bResult=NucUsb.EnableWinUsbDevice(); //Enable WinUsb driver
bResult=NucUsb.NUC_CheckFw(0); //Check device0 connect to PC
if(bResult==FALSE)
{
      AfxMessageBox(_T("Please reset device and Re-connect now !!!\n"));
return FALSE;
}
USHORT typeack=0x0;
/* Set SDRAM mode,others mode can refer to Serial.h */
bResult=NucUsb.NUC_SetType(0,SDRAM,(UCHAR *)&typeack,sizeof(typeack));
if(bResult==FALSE)
{
AfxMessageBox(_T("typeack failed !!!\n"));
NucUsb.NUC_CloseHandle();
```

```
return FALSE;
}

.....skips.....

/* Write SDRAM deader to device  */

bResult=NucUsb.NUC_WritePipe(0,(UCHAR *)lpBuffer,
sizeof(SDRAM_RAW_TYPEHEAD));
if(bResult==FALSE)
{
      delete []lpBuffer;
      NucUsb.NUC_CloseHandle();
      fclose(fp);
      AfxMessageBox(_T("Write SDRAM head error\n"));
      return FALSE;
}

/* waiting for devcie return ack, and check to receive sdram header */

bResult=NucUsb.NUC_ReadPipe(0,(UCHAR *)&ack,4);
if(bResult==FALSE)
{
      delete []lpBuffer;
      NucUsb.NUC_CloseHandle();
      fclose(fp);
      AfxMessageBox(_T("SDRAM head ack error\n"));
      return FALSE;
}
.....skips.....
while(scnt>0)
{
      fread(lpBuffer,BUF_SIZE,1,fp);
      /* Write data to device,length is 4096bytes */

      bResult=NucUsb.NUC_WritePipe(0,(UCHAR *)lpBuffer,BUF_SIZE);
      if(WaitForSingleObject(m_ExitEvent, 0) != WAIT_TIMEOUT)
      {
                  delete []lpBuffer;
                  fclose(fp);
                  return FALSE;
```

```
        }

        if(bResult==TRUE)
        {
                total+=BUF_SIZE;
                pos=(int)(((float)(((float)total/(float)file_len))*100));
                posstr.Format(_T("%d%%"),pos);
                /*Waiting for device return ack */
                bResult=NucUsb.NUC_ReadPipe(0,(UCHAR *)&ack,4);
                if(bResult==FALSE || ack!=BUF_SIZE)
                {
                        delete []lpBuffer;
                        NucUsb.NUC_CloseHandle();
                        fclose(fp);
                        AfxMessageBox(_T("ACK error !"));
                        return FALSE;
                }
.........skips.........
}
```

5.2 Firmware Code(xusb.bin)

NuWriter Firmware (xusb.bin)的開發環境為 Keil IDE。Keil要有支援 ARM9，需使用 MDK Plus 或是 Professional 版本。

Feature	MDK Edition			
	Professional	Plus	Essential	Lite
	All-in-one solution including Middleware	Supports all microcontroller cores and Middleware	Supports selected Cortex-M	Free with code size limit: 32 KBytes
Device Support				
Arm Cortex-M0/M0+/M3/M4/M7	✔	✔	✔	✔
Arm Cortex-M23/M33 Non-secure only	✔	✔	✔	✘
Arm Cortex-M23/M33 Secure and non-secure	✔	✔	✘	✘
Armv8-M Architecture Models including FastModel	✔	✘	✘	✘
Arm SecurCore®	✔	✔	✘	✘
Arm7™, Arm9™, Arm Cortex-R4	✔	✔	✘	✘

使用Keil可以開始NuWriterFW專案，開啟後如下圖示：

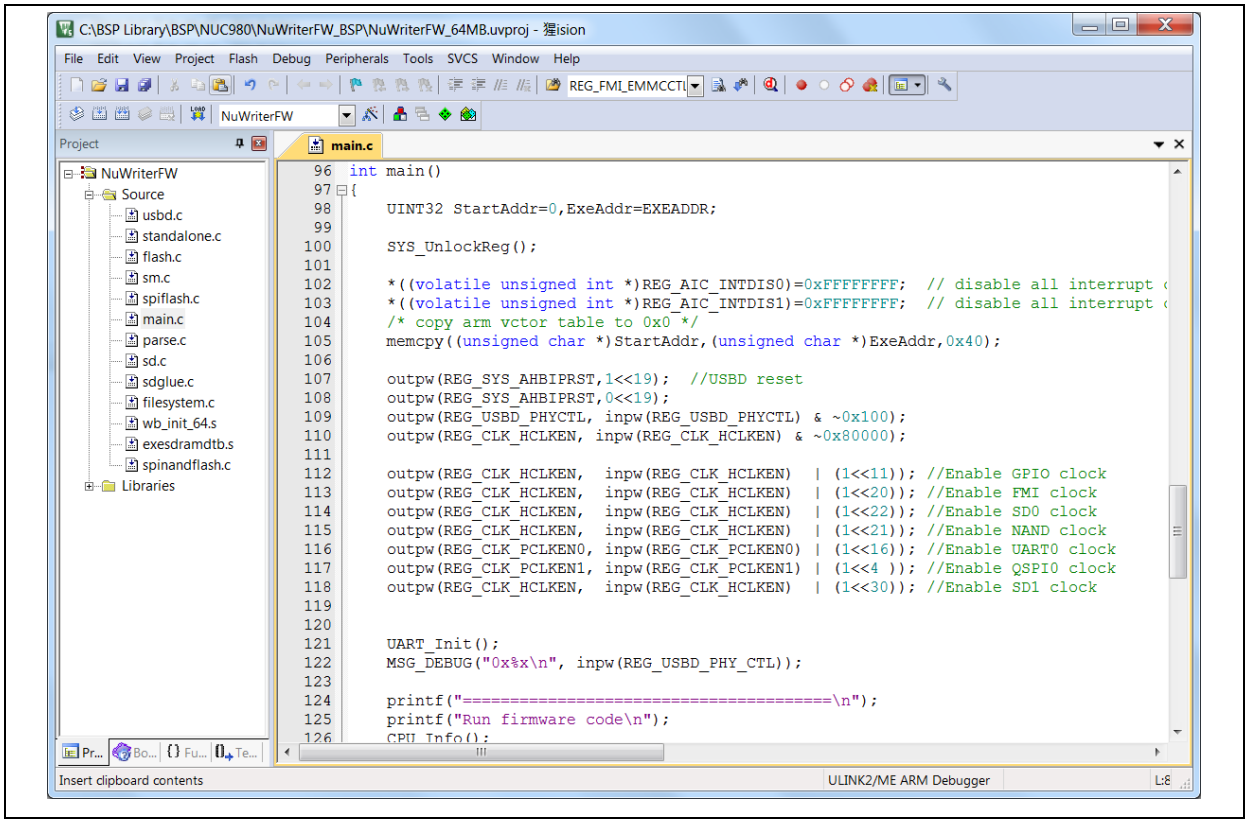


Figure 5-5 NuWriterFW – Project

Firmware code主要功能是透過NuWriter將xusb.bin 傳到設備並且執行，執行後將等待NuWriter透過USB傳送命令，根據命令作相對應的動作。程式執行後主要會while(1)在ParseFlashType(void)函數中等待USB傳送_usbd_flash_type，如下面表示，假設需要增加從NuWriter傳來新的命令，則並須增加_usbd_flash_type的值，以及需要修改ParseFlashType(void) 函數和usbd_control_packet(void)函數，可以參考下面程式中註解的說明。

```
INT ParseFlashType(void){
switch (_usbd_flash_type){
    /* Add new commands from nuwriter transmit */
    /*
    case yyyyy:
      TODO: Add your control notification handler code here
    break;
    */
    case USBD_FLASH_SDRAM: {
        UXmodem_SDRAM();
        _usbd_flash_type = -1;
    }
    break;
    case USBD_FLASH_MMC: {
        UXmodem_MMC();
        _usbd_flash_type = -1;
```

```
    }
    break;
    case USBD_FLASH_NAND: {
        UXmodem_NAND();
        _usbd_flash_type = -1;
    }
    break;
    case USBD_FLASH_SPI: {
        UXmodem_SPI();
        _usbd_flash_type = -1;
    }
    break;
    case USBD_FLASH_SPINAND: {
        UXmodem_SPINAND();
        _usbd_flash_type = -1;
    }
    break;
    case USBD_INFO: {
        UXmodem_INFO();
        _usbd_flash_type = -1;
    }
```

usbd.c中的usbd_control_packet()函數，如下：

```
void usbd_control_packet()
{
      ........skips........
      if (_usb_cmd_pkt.bRequest == 0xb0){
      /* Add new commands from nuwriter transmit */
      /*
      if (_usb_cmd_pkt.wValue == (xxxxx)){
            _usbd_flash_type = yyyyy;
            outpw(REG_USBD_CEP_CTRL_STAT, CEP_NAK_CLEAR);
            // clear nak so that sts stage is complete
      }
      xxxxx : NuWriter transmit value
      yyyyy : ParseFlashType(void) Parse value
      */

      if (_usb_cmd_pkt.wValue == (USBD_BURN_TYPE+USBD_FLASH_SDRAM)){
            _usbd_flash_type = USBD_FLASH_SDRAM;
            outpw(REG_USBD_CEP_CTRL_STAT, CEP_NAK_CLEAR);
```

```
            // clear nak so that sts stage is complete
      }
      else if (_usb_cmd_pkt.wValue == (USBD_BURN_TYPE+USBD_FLASH_NAND)){
            _usbd_flash_type = USBD_FLASH_NAND;
            // clear nak so that sts stage is complete
            outpw(REG_USBD_CEP_CTRL_STAT, CEP_NAK_CLEAR);
      }
      else if (_usb_cmd_pkt.wValue == (USBD_BURN_TYPE+USBD_FLASH_NAND_RAW))
      {     _usbd_flash_type = USBD_FLASH_NAND_RAW;
            // clear nak so that sts stage is complete
            outpw(REG_USBD_CEP_CTRL_STAT, CEP_NAK_CLEAR);
      }
      else if (_usb_cmd_pkt.wValue == (USBD_BURN_TYPE+USBD_FLASH_MMC)){
            _usbd_flash_type = USBD_FLASH_MMC;
            // clear nak so that sts stage is complete
            outpw(REG_USBD_CEP_CTRL_STAT, CEP_NAK_CLEAR);
      }
      ........skips........
}
```

FW程式修改後編譯成xusb.bin，如Figure 5-6所示。編譯後會自動將xusb.bin覆蓋至NuWriter資料夾，如Figure 5-7所示。

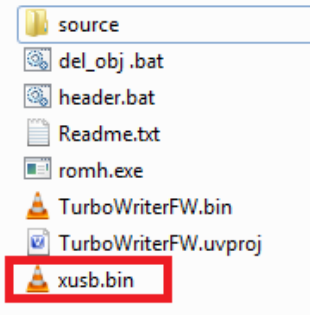


Figure 5-6 NuWriterFW – Folder

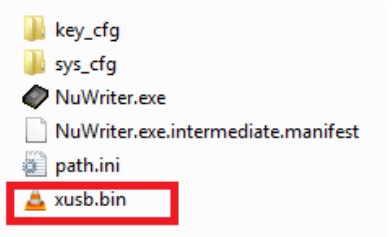


Figure 5-7 NuWriter – Folder

6 REVISION HISTORY

Date	Revision	Description
2018.09.17	1.00	初版
2019.01.07	1.01	修改格式並強化可讀性
2019.01.23	1.10	eMMC/SD 格式化:支援四個分割區
2019.02.22	1.11	Pack Mode新增eMMC/SD Partition Image型態
2019.04.22	1.12	編輯改變

Important Notice

Nuvoton Products are neither intended nor warranted for usage in systems or equipment, any malfunction or failure of which may cause loss of human life, bodily injury or severe property damage. Such applications are deemed, "Insecure Usage".

Insecure usage includes, but is not limited to: equipment for surgical implementation, atomic energy control instruments, airplane or spaceship instruments, the control or operation of dynamic, brake or safety systems designed for vehicular use, traffic signal instruments, all types of safety devices, and other applications intended to support or sustain life.

All Insecure Usage shall be made at customer's risk, and in the event that third parties lay claims to Nuvoton as a result of customer's Insecure Usage, customer shall indemnify the damages and liabilities thus incurred by Nuvoton.